Universidad
Carlos III de Madrid
www.uc3m.es

APLICACIÓN DEL PARADIGMA DE REDES CENTRADAS EN CONTENIDO A REDES DE SENSORES

Autor: David Carrillo Sánchez

Titulación: Grado en Ingeniería Telemática

Profesor: Ignacio Soto Campos

Fecha: 16 de Octubre de 2013

Tabla de Contenidos

1. Introducción

El presente trabajo de fin de grado va a exponer los fundamentos de un nuevo paradigma de comunicaciones en red, CCN (*Content Centric Networking*), una reciente propuesta alternativa al modelo TCP/IP actual. Dicho paradigma de comunicaciones se caracteriza por basar sus operaciones de comunicación en los propios nombres de los contenidos, abstrayéndose del concepto de localización de los contenidos. Es decir en CCN pedimos un contenido identificándolo por su nombre, en contraposición al modelo TCP/IP clásico en que, para pedir un contenido, tenemos que indicar la máquina que lo tiene (por su dirección IP o su nombre de dominio).

Por otro lado, el ámbito de aplicación del paradigma CCN en este proyecto son las redes inalámbricas de sensores WSN (*Wireless Sensor Networks*) caracterizadas por ser redes descentralizadas y por la limitación de recursos de sus nodos. La unión entre el paradigma CCN y las redes WSN se apoya en el hecho de que el principio de abstracción del concepto de localizaron en CCN es perfectamente compatible con el hecho de que en las redes WSN es más importante obtener determinada información mediante sensores que identificar el sensor particular que nos tiene que aportar dicha información.

Finalmente, la implementación práctica del proyecto se ha realizado empleando la plataforma Waspmote, un hardware orientado al despliegue de diferentes redes de sensores.

Para dar perspectiva acerca de la estructura del presente documento, en los apartados *2. Redes Centradas en Contenido* y *3. Redes Inalámbricas de Sensores*, se expondrán, respectivamente, los puntos clave y funcionamiento del proyecto CCNx [1][2] y de las redes WSN [3].

Una vez se ha descrito por separado CCN y WSN, en el apartado *4. Convergencia CCN-WSN* se explicará la forma en la que hemos diseñado el módulo CCN-WSN, describiendo entre otras cosas sus funciones básicas y la situación de las capas de estrategia y seguridad en la pila de protocolos.

La implementación del módulo CCN-WSN en la plataforma Waspmote [4] se describe en los apartados *5. Plataforma WASPMOTE* y *6. Prototipo de sistema CCN para Waspmotes*:

- Descripción técnica de los dispositivos Waspmote de Libelium.
- Implementación en C/C++ del módulo CCN-WSN.
- Pruebas del sistema y exposición de resultados.

Por último terminaremos en el capítulo 7 con las conclusiones y futuras líneas de trabajo que se abren a partir del proyecto.

2. Redes Centradas en Contenido

En el presente apartado se explicarán los puntos clave del paradigma CCN con el fin de que en la apartado *4. Convergencia CCN-WSN* puedan entenderse la decisiones en el diseño de nuestra solución.

A día de hoy, el modelo TCP/IP posee una posición dominante en Internet. Dicho modelo localiza las maquinas mediante direcciones IP y las aplicaciones finales a través de los números de puerto origen y destino.

Este modelo basado en "dónde" se encuentra un dato y hacia "dónde" ha de entregarse presenta dos grandes ineficiencias.

La mayor parte del trafico en Internet tiene su origen en la petición de una información a través de un nombre o referencia al contenido y no a través de una dirección IP.

- En la mayor parte de las aplicaciones en Internet los usuarios quieren un contenido y no les preocupa dónde está el contenido, por lo que tener que preocuparse por la dirección IP (o el nombre de dominio) de la máquina que tiene dicho contenido es solo un efecto secundario de cómo funciona TCP/IP.

- Incluso si dos nodos finales solicitan el mismo contenido, dicho contenido debe enviarse independientemente a cada uno de los nodos.

Por otro lado, las redes basadas en contenidos, o también llamadas *Named Data Networking*, *Content-Based Networking*, *Data-Oriented Networking* o *Information-Centric Networking*, y por supuesto *Content-Centric Networking* [5], basan todo la carga de seguridad y direccionamiento sobre los nombres de los contenidos. En la implementación de una red basada en contenidos hay que tener en cuenta 3 puntos claves [6]:

- **Persistencia.** El nombre asociado a un dato o servicio habrá de permanecer inalterable en la medida que el dato subyacente esté disponible. Es decir, que aunque el contenido sea modificado o cambie su ubicación, si conserva el nombre, ha de poder ser alcanzado.

- **Disponibilidad.** Una vez solicitado un determinado contenido, la red CCN ha de capaz de encontrar y reenviar la copia más cercana al solicitante manteniendo ciertos niveles de fiabilidad y latencia.

- **Autenticidad.** Dado que CCN se abstrae de conceptos cómo la identidad del origen de contenido, CCN ha de proporcionar mecanismos que aseguren que los contenidos no han sido manipulados.

En el caso de este proyecto, se han aplicado las premisas del proyecto CCNx de PARC (*Palo Alto Research Center*) [2], aunque existen múltiples vías de desarrollo y proyectos en lo que al paradigma CCN se refiere [7]. Todas ellas se caracterizan por tratar adaptarse o reutilizar los mecanismos ya integrados en Internet.

- **DONA (*Data-Oriented Network Architecture DONA*).** Esta vía de investigación busca rediseñar el servicio DNS (*Domain Names Service*) en Internet. De tal forma que no sea necesario modificar la capa IP subyacente [6].

- **TRIAD (*Translating Relaying Internet Architecture integrating Active Directories*).** TRIAD introduce el concepto de capa de contenido donde se concentrarían las operaciones de enrutamiento, almacenamiento y transformación de contenido. Debido a su compatibilidad con los protocolos de Internet existentes (TCP, IP, DNS...) y al hecho que resuelve ciertos problemas de NAT, TRIAD se presenta cómo una vía alternativa a Ipv6 [8].

- **ROFL (*Routing On Flat Labels*).** Propone un esquema de enrutamiento basado directamente etiquetas planas, unificando así, los diferentes los diferentes espacios de nombres [9].

2. 1. Terminología CCN

A lo largo de esta memoria, emplearemos los términos utilizados en la documentación técnica del proyecto CCNx [1] y, en la medida que sea posible, sin ser traducidos con el fin evitar confusión con otras fuentes bibliográficas o una traducción ambigua del término. Algunos de los términos más importantes son:

- *CCN (Content Centric Networking).* Se hará referencia al mismo cómo modelo CCN, paradigma CCN, proyecto CCN o enfoque CCN.

- *Interest.* Hace referencia a los mensajes de petición de contenido. Se harán referencia al mismo junto con términos tales cómo mensaje, paquete, petición o solicitud.

- *Data* y *ContentObject.* Ambos hacen referencia a los mensajes de repuesta de contenido. Se harán referencia a los mismos junto con términos tales cómo mensaje, paquete, contenido o respuesta.

- *ContentName* o simplemente *Name*. Se refiere al nombre de contenido, dato o fichero. (Ver apartado *2. 2. Nombres de Contenido*)

- *ContentStore o buffer.* Hacen referencia al lugar donde se almacenan los contenidos.

- Tabla PIT o PIT. Hacen referencia al lugar donde se almacenan las solicitudes de contenido pendientes.

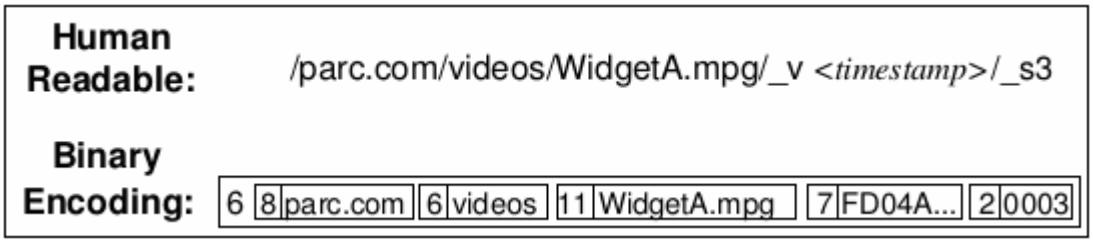
- Tabla FIB o FIB. Hacen referencia al lugar donde se almacenan las instrucciones para reenviar las solicitudes de contenido.
- *Face*. Es una generalización del término "interface" empleado en el modelo TCP/IP. Hace referencia al punto donde el proceso CCN se comunica con las aplicaciones y las redes.
- Consumidor y publicador de contenidos. Ambos términos hacen referencia a los nodos u aplicaciones dentro un nodo que solicitan contenidos o bien que proporcionan dichos contenidos.
- Reglas de Estrategia o Estrategias. Hacen referencia a un comportamiento específico del nodo a la hora de reenviar mensajes.

2. 2. Nombres de Contenido

Los nombres de contenido, que denominaré cómo *ContentName*, son la piedra angular del modelo CCN, apoyándose en los mismos para las operaciones de la capa de estrategia y la capa de seguridad. En el presente apartado, describimos la estructura e implicaciones de los *ContentName*.

En CCNx que es nuestro modelo de referencia, los *ContentName* siguen el formato URI (*Uniform Resource Identifier*) [10], es decir, una secuencia de componentes separados por el carácter "/" y en la que cada componente es una cadena de caracteres alfanuméricos de longitud variable y contenido arbitrario que carecen de un significado unitario en el dominio de nombres CCN. Esta libertad a la hora de formar el *ContentName* otorga a CCN un espacio de direccionamiento prácticamente infinito. Sin embargo, en la transmisión de los mensajes dependiendo de la implementación, el formato de ContentName podría ser una cadena caracteres o una codificación binaria indicando la longitud y el valor de cada componente.

Figura. 1 Formato binario-ASCII [11]



Por otro lado y de forma análoga a cómo una dirección IP define los conceptos parte de red y parte de host de la dirección, es recomendable que un *ContentName* se estructure jerárquicamente de izquierda a derecha, en 4 partes [12]. Para ilustrar esta estructura, pongamos, por ejemplo, el *ContentName "/uc3m.com/telemática/alumnos/video.mpg/v1/s2/0x92a2b403"*

- **Prefijo globalmente enrutable.** Aunque CCN no está relacionado con DNS, es muy recomendable utilizar la estructura y espacio de nombres de DNS para identificar al origen del contenido y que también proporciona una forma de crear contenidos de nombre único en cada dominio de Internet. Haciendo referencia al ejemplo anterior, el prefijo globalmente enrutable es *"/uc3m.com/..."*. Por lado, la libertad a la definir el ContentName y su estructura jerárquica es tal que incluso sería posible utilizar la información de un socket IP, por ejemplo *"/80.201.123.5:8080/..."*, dentro del *ContentName*. Este hecho parece contradecir el principio de deslocalización de contenidos de CCNx, pero no hay que olvidar que con una petición de contenido con el prefijo "/80.201.123.5:8080/..." no tiene por que traducirse en un segmento o datagrama con dirección de destino 80.201.123.5 y puerto de destino 8080.

- **Estructura organizativa.** Sería recomendable buscar la análoga a un sistema ficheros Unix, permitiendo identificar de forma única contenidos gracias a la estructura jerárquica de un sistema de fichero. No obstante, desde el punto de vista de CCNx, esta parte no representa la localización de un fichero dentro de una maquina. Por ejemplo, la secuencia de componentes "*.../telemática/alumnos/video.mpg/...*" no implica obligatoriamente que el contenido se encuentre en fichero "video.mpg" que se encuentra en el directorio "*/telemática/alumnos"*.

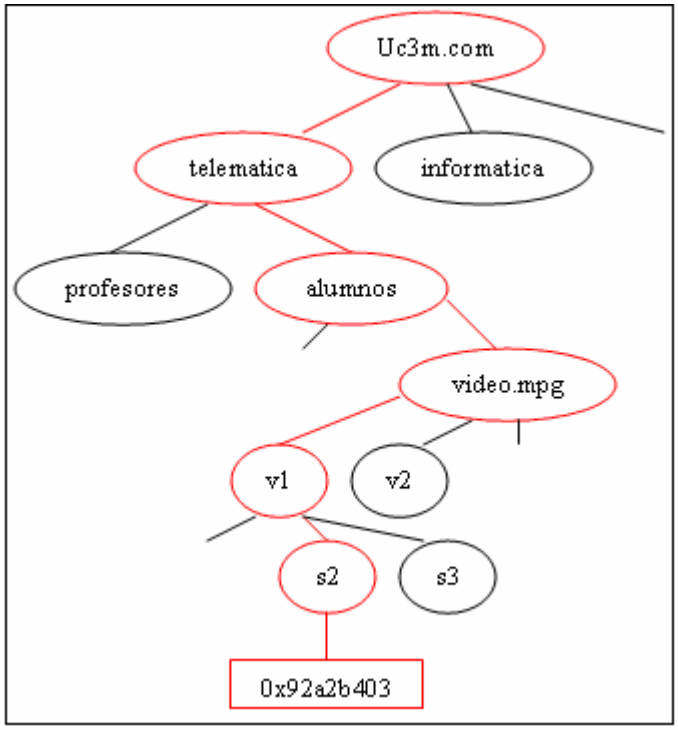
- **Marcadores.** Este tipo de componentes de ContentName sirven para especificar la versión, el segmento del contenido o, incluso, comandos que puedan ser interpretados por el módulo CCN. Más adelante, se explicarán los tipos y funcionamiento de los marcadores. Con respeto al ejemplo anterior, los marcadores se corresponden al fragmento del *ContentName "...v1/s2/..."* indicando que se trata de la versión 1 del contenido y del fragmento 2 de la versión 1.

- Componente implícito SHA256. Utilizado para verificar la integridad y autenticidad del conjunto nombre y contenido. En el ejemplo, "*.../0x92a2b403"*.

Figura. 2. Ejemplo de un ContentName

El resultado implícito de este formato es la creación de una estructura jerárquica en árbol que permite la identificaron inequívoca de un contenido.

Figura. 3. Estructura jerárquica de un ContentName



Adicionalmente, es necesario comentar ciertos aspectos de los *ContentName*:

A) Marcadores [12]

Tal y cómo hemos visto antes, los marcadores son un tipo de componentes del *ContentName* que CCN emplea para manejar la información generada dinámicamente o divisible en partes con sentido propio, por ejemplo, las secuencias de vídeo en tiempo real. Actualmente, el proyecto CCNx define 3 tipos de marcadores.

- **Versión (0xFD).** Usado por el publicador para indicar la versión de contenido usando la marca de tiempo en segundos de cuando fue creado. Se codifica insertando el byte 0xFD seguido de la marca de tiempo.
- **Segmentación (0x00 o 0xFB).** Identifica los fragmentos empleando la menor la cantidad de bytes posible, por ejemplo, el segmento 1 es codificado cómo 0x01 y el segmento 257 cómo 0x0101. Hay dos posibles formas de enumeración de fragmentos
    - Consecutiva. Se codifica insertando el byte 0x00 seguido del número de segmento.
    - No consecutiva. Se codifica insertando el byte 0xFB seguido del desplazamiento en bytes del segmento.
- **Comando extensible (0xC1).** Se utiliza para codificar comandos, solicitudes e identificadores especiales de un espacio de nombres. Los comandos siguen el siguiente formato: byte 0xC1, seguido del espacio de nombres, seguido del comando y, dependiendo del caso, los argumentos separados por el carácter "~". Por ejemplo, "/C1.org.ccnx.frobnicate~1~37" indica que se desea realizar la operación frobnicate del espacio de nombres "org.ccnx" pasando cómo parámetros los valores 1 y 37.

B) Orden Canónico [13]

Tal y cómo se muestra en las figuras 2 y 3, un *ContentName* es una secuencia de componentes que crea una estructura jerárquica en árbol. Sin embargo, puede darse el caso que un mismo *Interest* pueda ser satisfecho por más de un contenido del *ContentStore*.

En tal caso, se define el orden canónico cómo una función de comparación de secuencias de bytes, ordenándolas de forma equivalente la función mencmp [14]. Para ilustrar su funcionamiento, pongamos, por ejemplo, que tenemos 4 *ContentNames* en un *ContentStore* (Ver apartado *2. 4. 1. Content Store (buffer memory)*).

- A: /uc3m.com/telemática/alumnos/
- B: /uc3m.com/telemática/alumnos/video.mpg/v1/s2
- C: /uc3m.com/telemática/alumnos/video.mpg/v2/s1
- D: /uc3m.com/

La lógica detrás del orden canónico consiste en dos pasos. Primero, si X tiene una longitud menor que la longitud de Y, entonces X se encontrará antes que Y. Segundo, si X e Y tienen la misma longitud, se comparan byte a byte, tal que si X e Y difieren en el byte n y X[n] menor que Y[n], entonces X se encontrará antes que Y. En conclusión, el orden canónico para el ejemplo anterior sería, izquierda a derecha, D-A-B-C.

C) Prefijos del nombre

En general, un prefijo de CCN representa una colección de datos, por ejemplo si existen los nombres *"/uc3m.com/telemática/pdf"* y *"/uc3m.com/telemática/videos/leccion1.mpg"*, *"/uc3m.com/telemática"* es un prefijo de ambos *ContentName*. Aunque más adelante se explicarán en detalle, los usos de los prefijos CCN son:

- Identificar de los contenidos que se desea obtener.
- Tomar decisiones acerca del reenvió de las peticiones y las respuestas del contenido.

D) Componente implícito SHA256 [15]

En la implementación de CCN, se genera un componente SHA256 a partir del *ContentObject*. Este componente se añade al final del *ContentName* del mensaje Interest y es útil para asegurar que el *ContentObject* recibido es aquel que fue solicitado. No obstante, puede resultar contradictorio que esta componente represente un ContentObject que esta siendo solicita en el mensaje Interest. En los casos en los que el consumidor no tenga constancia del ContentObject a recibir, esta componente se genera a partir del ContentName.

Más adelante en el apartado *2. 5. 2. 3. Algoritmos de componente implícita*, explicaremos más en detalle este mecanismo.

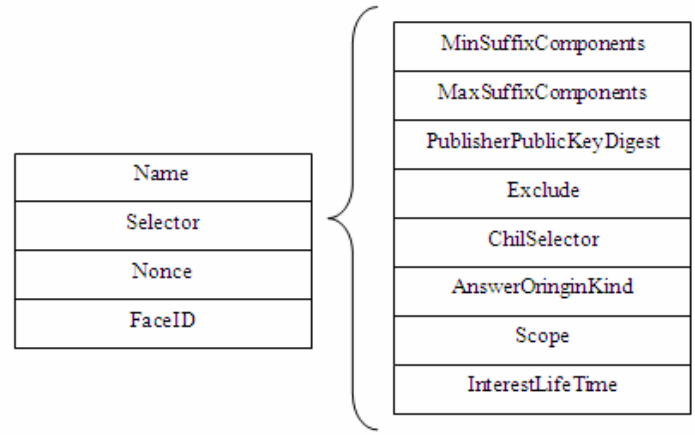
2. 3. Mensajes de CCN

CCN emplea dos tipos de mensajes: peticiones *Interest* y respuestas *Data* (también denominadas *ContentObject o Content*). En cuanto al formato, la posición y la longitud de ambos mensajes, no están restringidos por el proyecto CCNx.

2. 3. 1. Mensaje Interest

Realiza la solicitud de un contenido a través del *ContentName*. En la implementación más básica de CCN, el *Interest* es únicamente el nombre del contenido y el resto son campos opcionales. No obstante, se explicarán todos los campos definidos por el proyecto CCNx para poder entender más adelante las razones por las que algunos los excluimos y otros no, de nuestro diseño final en el proyecto [16].

Figura. 4. Paquete Interest CCN



- *Name.* También llamado *ContentName*. Almacena el nombre del contenido deseado. En el caso de un mensaje *Interest*, el nombre solicitado es considerado cómo un prefijo que puede ser más o menos especifico.
- *Selector [Opcional].* Contiene una serie de campos anidados que alteran que *ContentObject* será entregado en respuesta al *Interest*.
  - *MinSuffixComponents y MaxSuffixComponents [Opcionales].* Establece el número mínimo y máximo de componentes del *ContentName* ajustando las búsquedas de más genéricas a más específicas. Por defecto, sus valores son 0 e infinito, respectivamente.
  - *PublisherPublicKeyDigest [Opcional].* El valor almacenado en este campo ha de coincidir con el *PublisherPublicKeyDigest* del *ContentObject* solicitado. Dicho valor es un SHA-256 digest generado a partir de una clave pública. (Ver apartado *2. 5. 2. 3. Algoritmos de componente implícita.*)
  - *Exclude [Opcional].* Describe una secuencia de listados de componentes que deberían ser excluidas a continuación del prefijo del nombre cuando se responda al *Interest*. Dichos listados de componentes habrán seguir el orden canónico.

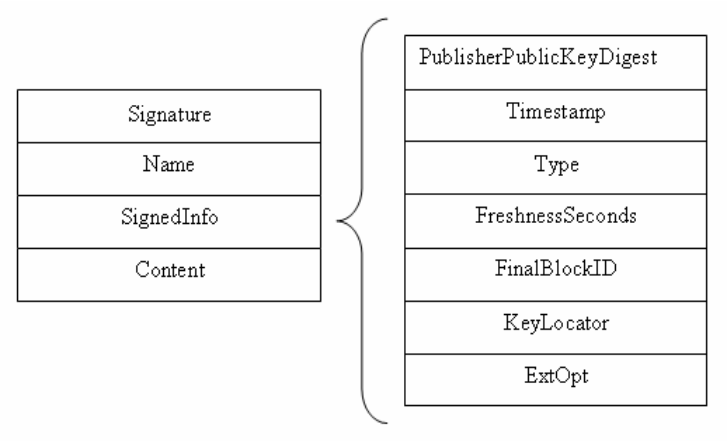
- *ChildSelector [Opcional].* Estable una preferencia para seleccionar el *ContentObject.* Si valor es 0, se prefiere el ContentObject más a la izquierda, siguiendo el orden canónico de CCNx. Si tiene valor uno, se prefiere el ContentObject más a la derecha. Por ejemplo, si se seleccionara el *ContentObject* más a la derecha junto con los marcadores versión y segmentación (ver apartado *2. 2 Nombre de Contenido, Marcadores*) aumentaría la probabilidad de obtener el primer segmento de la última versión de un contenido.
- *AnswerOringinKind [Opcional].* Codifica ciertas opciones que pueden alterar el origen del *ContentObject.*
  - 0 = "No responder desde un *ContentStore*".
  - 1 = "Responder desde el existente *ContentStore*"
  - 2 = "La respuesta podría ser generada cómo nueva"
  - 3 = "Uso normal del *ContentStore*". Opción por defecto.
  - 4 = "La respuesta podría estar anticuada". (Ver apartado *2. 3. 3. Mecanismos auxiliares, Marca de antigüedad.*)
  - 16 = "Marcar respuesta cómo anticuada en el *ContentStore*".
- *Scope [Opcional].* Indica hasta qué punto se puede propagar un *Interest.* En ningún caso se trata de un mecanismo que marca la durabilidad del paquete en tiempo o número de saltos cómo el TTL (*Time To Life*) de IPv4. Los valores definidos para este campo son:
  - 0 = Limita la propagación al proceso CCN. Evitando que alcance a otras aplicaciones dentro del nodo o a otros nodos CCN.
  - 1 = Limita la propagación a las aplicaciones dentro del nodo CCN. Evitando que alcance a otros nodos CCN.
  - 2 = Permite la propagación más allá del propio nodo y sus aplicaciones.
- *InterestLifetime [Opcional].* Indica el tiempo que la petición de *Interest* se guardará en el nodo actual.

- *Nonce [Opcional].* Contiene una marca única y aleatoria para cada mensaje *Interest*, creada para detectar y eliminar las peticiones duplicadas y suprimir los bucles de mensajería.
- *FaceID.* Está pensado para indicar un *Face* de salida preferente durante el proceso de reenvío del *Interest*.

2. 3. 2. Mensaje ContentObject

Se trata del paquete que transporta los datos en repuesta a una petición *Interest* [17]. La unión de nombre de contenido, firma de publicador y carga útil de datos convierte los *ContentObject* en unidades de datos idempotentes, auto-identificables y auto-autenticables, lo cual implica cada paquete es potencialmente útil a muchos consumidores del mismo contenido [11].

Figura. 5. Paquete ContentObject de CCN



- *Signature.* Se calcula a partir del conjunto *Name, SignedInfo* y contenido por parte del publicador del contenido aplicando un cierto algoritmo de cifrado, generalmente SHA256, y asegurando así, que no se han producido manipulaciones ni errores durante la transmisión. (Ver apartados *2. 3. 3. Mecanismos auxiliares, Firma digital* y *2. 5. 2. 1. Algoritmos de la firma digital*)
- *Name.* También llamado *ContentName*. Almacena el nombre del contenido de forma específica.
- *SignedInfo.* Contiene diversos sub-campos para el manejo del contenido.
    - *PublisherPublicKeyDigest.* Identifica al publicador que firmo el contenido. El valor es un SHA-256 digest generado a partir de una clave pública (ver apartado *2. 5. 2. 3. Algoritmos de componente implícita.*)
    - *Timestamp.* Almacena cuándo fue creado el *ContentObject* (ver apartado *2. 3. 3. Mecanismos auxiliares, Marca de tiempo*).
    - *Type [Opcional].* Típicamente se marca un *ContentObject* cómo *DATA*, aunque es posible codificar 3 bytes de valores entre los que se encuentran.
        - DATA (0x0C04C0). Tipo de *ContentObject* por defecto, cuando este campo es omitido.
        - ENCR (0x10D091). El contenido esta encriptado.

- GONE (0x18E344). Marcador.
- KEY (0x28463F). Clave publica.
- LINK (0x2C834A). Enlace.
- NACK(0x34008A). El contenido no está disponible en ese momento.

- *FreshnessSeconds [Opcional]*. Indica el tiempo en segundos, que tras la llegada del contenido el nodo CCN debería esperar antes de marcar el contenido cómo anticuado (ver apartados *2. 3. 3. Mecanismos auxiliares, Marca de antigüedad y 2. 4. 1. Content Store (buffer memory)*).
- *FinalBlockID [Opcional]*. Identifica el último bloque de una secuencia de fragmentos. El valor de este campo debería ser el *ContentName* del último fragmento.
- *KeyLocator [Opcional]*. Indica dónde encontrar la clave para verificar el contenido. (Ver apartado *2. 5. 2. 4. Gestión de claves públicas)*
- *ExtOpt [Opcional]*. Almacena meta-datos adicionales, es decir, datos que describen otros datos que pudieran ser utilizados por elementos de procesamiento del propio nodo. Dichos meta-datos no han de interferir con el resto de la estructura del mensaje. Dado que se no define una estructura o formato especifico, CCNx recomienda la omisión de este campo.

- *Content*. Carga de datos finales.

2. 3. 3. Mecanismos auxiliares

A lo largo de la descripción de los mensajes CCNx, se ha hecho referencia a ciertos conceptos relacionados con CCN que requieren una descripción aparte.

A) Marca de antigüedad (Staleness) [18]

Dentro de *ContentStore*, se asocia un bit de antigüedad a cada *ContentObject*. Obviamente, a la llegada de un *ContentObject*, ya sea nuevo o preexistente en el *ContentStore*, este bit es marcado a 0 y es activado por alguna de las siguientes vías:

- Explícitamente por parte del propio nodo, empleando un mensaje Interest con la opción "Marcar respuesta cómo anticuada en el *ContentStore*" del campo *AnswerOriginKind* y limitando el alcance al propio módulo CCN del nodo (*Scope*=0).
- Por tiempo, desde la llegada del *ContentObject* y transcurrido el tiempo del campo *FreshnessSeconds*.

Normalmente cuando este bit está activo, el *ContentObject* no será entregado en respuesta a *Interest* a menos que la opción "La respuesta podría estar anticuada" (0x4) esté presente en el campo *AnswerOriginKind* del mensaje *Interest*.

Por otro lado y considerando un espacio del *ContentStore* limitado, este bit puede ser usado para priorizar la eliminación de un *ContentObject* obsoleto y liberar memoria.

B) Marca de tiempo (*Timestamp*) [19]

El concepto tiempo en CCN, toma cómo referencia el tiempo en los sistemas UNIX, que mide el tiempo en segundos desde el 1 de Enero de 1970 a las 00:00AM. Dos puntos clave a la hora de usar marcas de tiempo son longitud en bits de la marca de tiempo y las unidades de tiempo que representan. Por ejemplo, una marca de tiempo de 32 bits medida en segundos tiene cómo fecha máxima de utilización el 7 de Febrero de 2106.

En el caso de CCN, se podría establecer una unidad de tiempo de 2^{-12} segundos (aproximadamente 0.25 milisegundos) y una marca de tiempo de 48bits (de los cuales 12bits definen la precisión) y teniendo cómo fecha máxima de utilización el 20 de Agosto de 4147.

C) Firma digital (*Signature*) [20]

Todo *ContentObject* contiene una firma digital que certifica la identidad del publicador y la integridad del contenido. Esta firma se construye a partir del *ContentName*, *SignedInfo* y el contenido en sí mismo, aplicando un determinado algoritmo criptográfico. A su vez, la firma digital se compone de 3 partes:

- *DigestAlgorithm.* Indica el algoritmo utilizado en la generación de la firma digital. El valor es este campo es el valor OID (*Object Identifier*) para dicho algoritmo. En general, algoritmo utilizado será SHA256 cuyo OID es "2.16.840.1.101.3.4.2.1".
- *Witness.* Información adicional para la verificación del *ContentObject*. Cuando una firma digital es generada para múltiples *ContentObject*, el presente campo almacena la información necesaria para la verificación individual de cada *ContentObject*.
- *SignatureBits*. Contiene la firma digital en sí misma.

2. 4. Estructura de un nodo CCN

El nodo CCN ha de ser capaz de manejar la información relativa a los mensajes de *Interest* y *Data* y redirigirla a los *Faces* o tablas correspondientes. Para ello, un nodo se compone de las siguientes tres tablas y tantos *Face* cómo sean necesarios [11].

- ***Content Store (buffer memory).***
- ***PIT (Pending Interest Table).***
- ***FIB (Forwarding Information Base).***

Al igual que ocurría con el formato de los mensajes, la información que se almacena en las tablas puede variar de una implementación a otra. Por tanto, las estructuras de datos que representan cada tabla se describirán en el diseño final (ver *Apendice I*).

2. 4. 1. Content Store (buffer memory)

Actúa cómo un *buffer* de memoria de paquetes de datos. Simplificando al máximo su funcionamiento, cuando un *ContentObject* llega al nodo CCNx, este es almacenado o renueva uno ya existente y cuando hay una coincidencia con una petición *Interest*, se realiza una copia y envía por la *Face* de entrada del *Interest*. Las entradas en el *ContentStore* almacenan un *ContentName* específico, al contrario que las entradas de las tablas PIT y FIB cuyos *ContentName* asociados representan prefijos.

Una de las consideraciones más importantes en el diseño del *ContentStore*, son las reglas que definen la persistencia de datos. A diferencia de un *buffer* en un nodo IP, un paquete de datos en CCN no se elimina una vez es enviado, sino que podrían ser parcialmente persistentes aplicando políticas de reemplazo que maximicen la probabilidad de compartir los contenidos. Estas políticas de reemplazo podrían ser [21]:

- *Least Recently Used (LRU).* Esta política de reemplazo busca descartar aquellos contenidos que no han sido solicitados de forma reciente. Para implementar este mecanismo, habría que añadir una marca de tiempo asociada al *ContentObject* que indicase la última vez que fue solicitado mediante un *Interest*.
- *Least Frequently Used (LFU).* En este caso, se busca descartar aquellos contenidos que hayan sido solicitadas en un menor número de ocasiones. Para implementar este mecanismo, habría que añadir un contador de solicitudes *Interest* asociada al *ContentObject* que indicase el número de ocasiones que fue solicitado mediante un *Interest.*

Esta capacidad de compartir paquetes reduce la demanda de ancho de banda de carga y la latencia de descarga.

No obstante y de forma paralela a la gestión de la memoria escogida, no hay que olvidar la implementación de la marca de antigüedad (ver apartado *2. 3. 3. Mecanismos auxiliares, Marca de antigüedad*) cuyo estado priorizaría el descarte de uno u otro *ContentObject*.

2. 4. 2. PIT (Pending Interest Table)

Mantiene un registro de los paquetes de *Interest* enviados hacia las potenciales fuentes de datos. De tal forma que, los paquetes *ContentObject* puedan ser retornados a los nodos que los solicitaron. En cuanto un nodo recibe un *ContentObject* previamente solicitado, elimina la entrada asociada ese prefijo en la tabla PIT. En la implementación básica de esta tabla PIT, hay que considerar dos puntos clave:

- Frecuencia en el reenvío de los mensajes *Interest* pendientes.
- El tiempo de vida de los mensajes *Interest* en la tabla PIT.

En apartado *2. 3. 1. Mensaje Interest,* se mencionó la existencia de dos campos en el mensaje *Interest* que afectan directamente a las entradas de la tabla PIT, *InterestLifetime* y *Nonce*. En una implementaron más sofisticada que requiriera el uso de estos campos, ha de tenerse en cuenta los detalles del funcionamiento de la tabla PIT. Para ello pongamos el siguiente ejemplo.

- Un nodo CCN recibe un *Interest* a través del *Face* 1 solicitando el prefijo *"/uc3m/telemática"*.
- Al no poder ser satisfecho por el *ContentStore*, se genera una entrada en la tabla PIT con el prefijo solicitado (*"/uc3m/telemática"*) y el *Face* de entrada (*Face* 1).
- Instantes después llega otro *Interest* a través del *Face* 2 solicitando el mismo prefijo, por lo cual se modificaría la entrada en la tabla PIT para indicar que el prefijo *"/uc3m/telemática"* fue solicitado a través del *Face* 1 y del *Face* 2.

En conclusión, la información de las entradas en tabla PIT son parcialmente sobrescritas cuando el prefijo solicitado coincide con dos o más mensajes *Interest* recibidos. Dado que probablemente la información contenida en los campos *InterestLifetime* y *Nonce* sea diferente en cada mensaje *Interest*, una solución podría ser generar una entrada en la tabla PIT única por cada mensaje *Interest*.

2. 4. 3. FIB (Forwarding Information Base)

Se usa para reenviar los paquetes *Interest* hacia la potenciales fuentes de datos. A diferencia de la tabla FIB de una tabla de reenvío IP, CCN permite enviar el mismo *Interest* por múltiples *Face* de salida. Esto permite tener múltiples fuentes de datos solicitadas en paralelo.

Dado que la complejidad de una tabla FIB CNN es mayor, es necesario un conjunto de instrucciones especiales que definidas forman las reglas de estrategia. (Ver apartado *2. 5 .1. 3. Reglas de estrategia y protocolos auxiliares*.).

2. 4. 4. Procesamientos de mensajes

En el presente apartado, describimos respectivamente el procesamiento estándar de los mensajes *Interest* y *ContentObject [22]*. En dicha descripción, no se tiene en cuenta los mecanismos de verificación o los campo opcionales descritos en los apartados *2. 3. 1. Mensaje Interest* y *2. 3. 2. Mensaje ContentObject*.

A) Procesamiento del mensaje *Interest*

A la llegada de mensaje *Interest*, su *ContentName* y el *Face* de entrada del mensaje son procesados en la siguiente secuencia pasos.

1. ContentStore. Si hay coincidencia en el *ContentStore* con el *ContentName* solicitado, se envía una copia del *ContentObject* correspondiente por el *Face* entrada. Sino hay ninguna coincidencia, el proceso continúa en la tabla PIT.

2. PIT. Primero comprueba el *ContentName* y, según el caso, la lista de Faces entrada.

- Si hay coincidencia en la tabla PIT con el *ContentName* solicitado, se comprueba la lista de Faces asociada a la entrada de la tabla.
    - Si el *Face* de entrada se encuentra en la lista de *Faces* solicitantes, ello implica que el *Interest* ya fue solicitado por dicho *Face* y simplemente se descarta el mensaje *Interest*.
    - Si el *Face* de entrada no se encuentra en la lista de *Faces* solicitantes, se añade la *Face* a la lista y el proceso continua en la tabla FIB.
- Si no hay coincidencia en la tabla PIT con el *ContentName*, se añade una nueva entrada en la tabla PIT con el *ContentName* y la *Face* de entrada en la lista de Faces asociada y el proceso continua en la tabla FIB.

3. FIB. Aplicando el criterio de mayor coincidencia (equivalente al criterio *Longest Prefix Match* de IP), se reenvía el *Interest* en cada una de las *Faces* de la lista. En el caso de no haber ninguna coincidencia en la tabla FIB y dependiendo de la reglas de estrategia impuestas, el mensaje *Interest* podría ser descartado o enviado por un *Face* por defecto o por todas la Faces disponibles.

B) Procesamiento del paquete *ContentObject*

A la llegada de un mensaje *ContentObject*, el paquete es procesado en los siguientes pasos.

1. ContentStore. Tanto si una hay coincidencia con el *ContentName* como si no, se sobrescribe o se añade el *ContentObject* de la tabla, marcando el bit de antigüedad a 0 y los valores del campo *TimeStamp* y del campo *PublisherPublicKeyDigest*. Una vez creada la entrada en el *ContentStore*, el proceso en la tabla PIT.

2. PIT. Si hay coincidencia con el *ContentName*, el *ContentObject* se envía por cada Faces de la lista asociada y se elimina la entrada la tabla *PIT*.

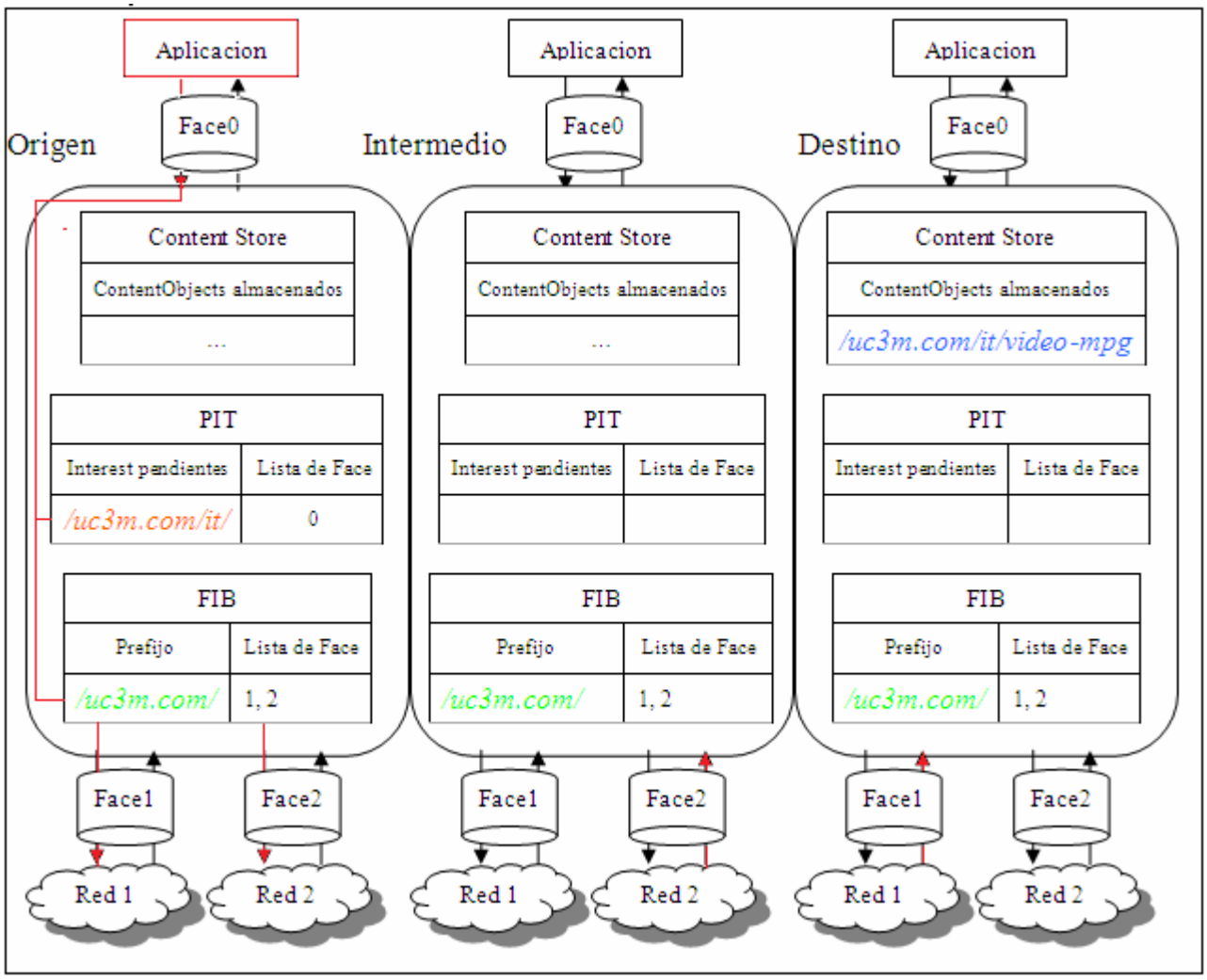
Para ilustrar mejor el papel de cada uno de los elementos que componen un nodo CCN y partiendo de un ejemplo simple, se describirá el procesamiento de la petición y entrega del contenido *"/uc3m.com/it/video-mpg"* en una red CCN compuesta por 3 nodos idénticos, a los que denominaremos "Origen", "Intermedio" y "Destino".

1. Partimos de una situación inicial en la que el nodo Destino posee en su *ContentStore* una entrada para *"/uc3m.com/it/video-mpg"* y que todos los nodos poseen una entrada en sus tablas FIB que indica que el prefijo *"/uc3m.com/"* se haya a través de las *Face* 1 y *Face* 2.

2. Una aplicación de nodo Origen genera un *Interest* por *"/uc3m.com/it/video-mpg"*, registra la petición en la PIT asociada a la *Face* 0 y envía la petición siguiendo la tabla FIB.

3. El *Interest* alcanza el nodo Intermedio a través de la *Face* 2 y el nodo Destino a través de la *Face* 1, por ejemplo.

Figura. 6. Pasos del ejemplo 1, 2 y 3

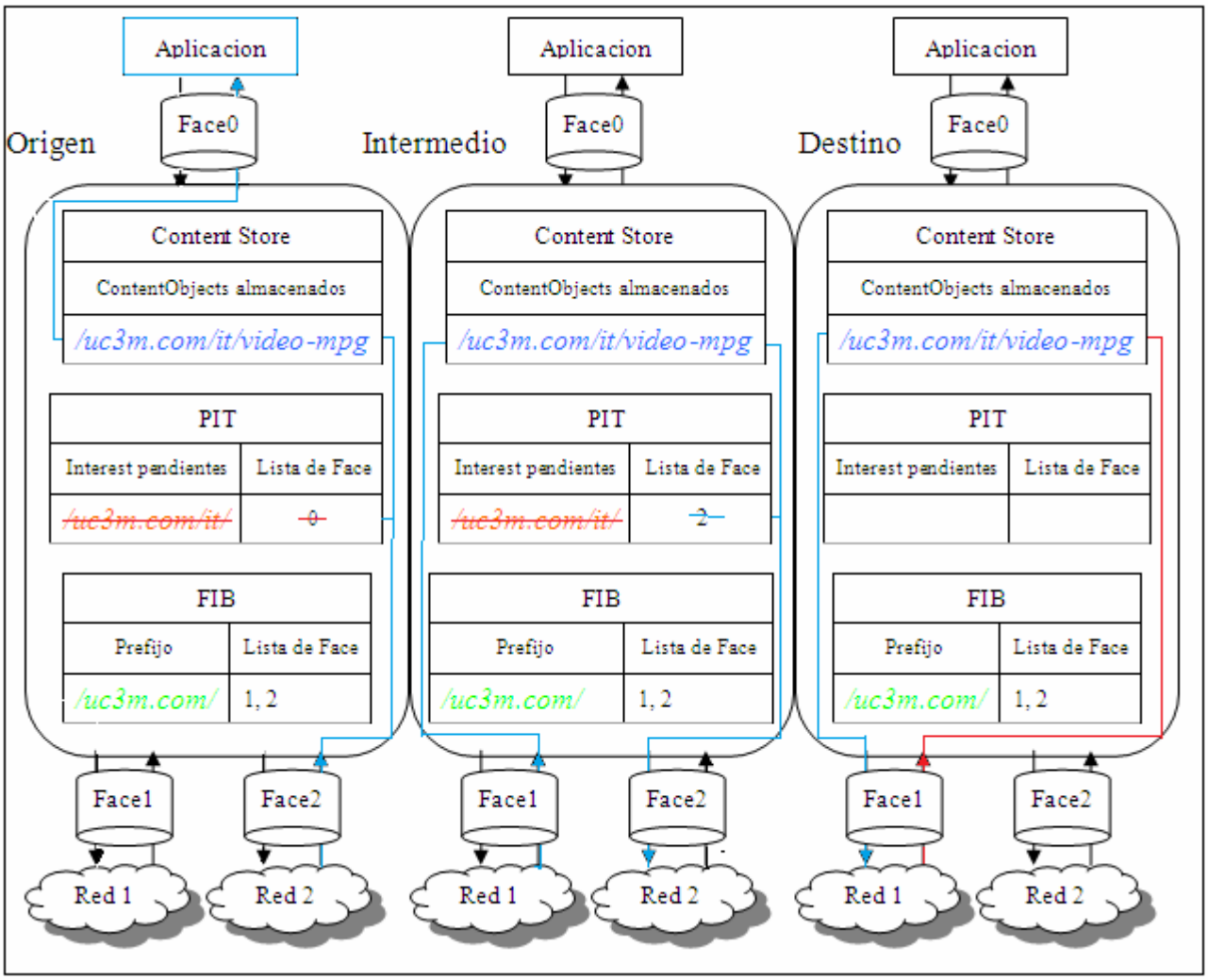


4. El nodo Destino extrae el *ContentObject* de su *ContentStore* y lo envía por la *Face* desde la cual llego el mensaje *Interest* correspondiente, en este caso, a la Face 1.

5. El paquete *ContentObject* alcanza el nodo Intermedio, se inserta una entrada en el *ContentStore* y se lee la tabla PIT para conocer las Faces de salida. En este caso, el *ContentObject* es enviado por la *Face* 2 y, dado que es la última *Face* de la lista, se elimina la entrada en la PIT.

6. Finalmente, el paquete *ContentObject* alcanza el nodo Origen, se inserta una entrada en el *ContentStore* y se envía el dato por la *Face* 0 leyendo la tabla PIT. En cuanto al estado final de las tablas de cada uno de los nodos, cada *ContentStore* almacena una copia del *ContentObject* para entregar a futuras peticiones y cada tabla PIT está vacía ya que todos los mensajes *Interest* pendientes han sido satisfechos.

Figura. 7. Pasos del ejemplo 4, 5 y 6

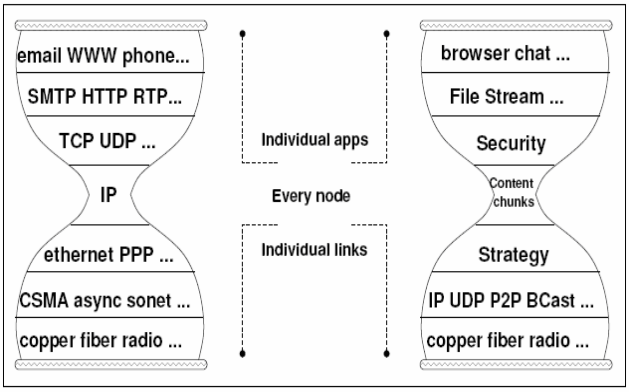


2. 5. Modelo de Protocolo

Un modelo de protocolo cómo TCP/IP divide las competencias y funciones de la comunicación en capas de protocolo. Tomando cómo referencia una de estas capas, se desarrollan protocolos que solo pueden ser interpretados dentro del mismo nivel de la pila. [12].

En el caso de CCN, no puede ser completamente ajeno al modelo TCP/IP, ni tampoco considerarse un nuevo protocolo de una capa concreta o una nueva capa de protocolos añadida a la pila. La forma más apropiada de visualizar una implementación de CCN es la de un *middleware* universal que actúe cómo intermediario entre las aplicaciones (capa de aplicación en el modelo TCP/IP) y las redes (capas de transporte, Internet y acceso a red en el modelo TCP/IP).

Figura. 8. Stack TCP/IP y CCN [11]

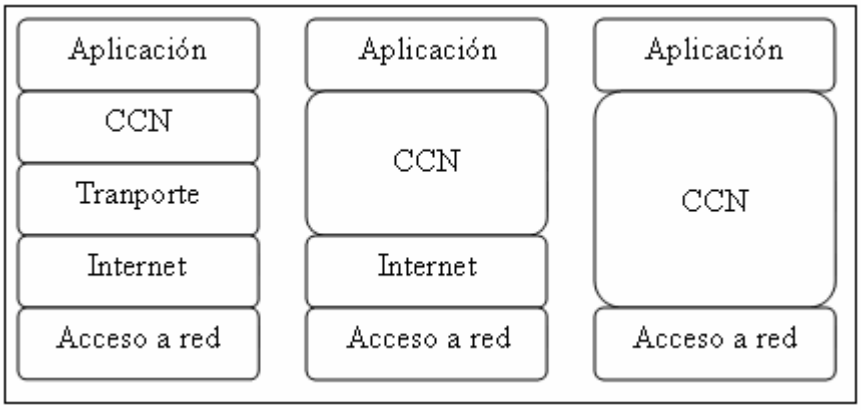


En referencia a la figura anterior, dentro de un módulo CNN pueden distinguir tres capas.

- Capa de seguridad.
- Fragmento de contenido.
- Capa de estrategia.

Por otro lado y desde el punto de vista de TCP/IP, es posible apoyar el módulo CCN directamente sobre la capa de transporte, la capa de Internet o la capa de acceso a red absorbiendo las funciones de las capas superiores.

Figura. 9. Ubicaciones del módulo CCN en el stack TCP/IP



2. 5. 1. Capa de Estrategia

La capa de estrategia es un conjunto de instrucciones, atributos y disparadores encargados de obtener un dato a partir de un prefijo [11] mediante la creación de reglas de estrategia. Esta definición abarca aspectos relativos a la capa de transporte o la capa de Internet del modelo TCP/IP.

2. 5. 1. 1. Funciones relativas a la Capa de Transporte

CCN está diseñado para operar sobre un servicio de entrega no fiable con conectividad dinámica y móvil, por lo que en la práctica, apoyar el módulo CCN sobre un socket UDP/IP es la opción más razonable. No obstante, la comparativa con el protocolo TCP ilustrará mejor las características CCN.

A) Secuenciación de mensajes

En el protocolo de transporte TCP, los segmentos de datos son identificados mediante números de secuencia. Estos números de secuencia representan posiciones relativas en bytes de un bloque de datos y son usados para tareas de confirmación, retransmisión y entrega en orden de los segmentos.

Aunque en otros aspectos CCN es más simple que TCP, en este caso CCN emplea un mecanismo más sofisticado ya que CCN podría enviar o solicitar múltiples porciones de datos a múltiples nodos CCN.

La forma en la que CCN puede manejar pequeñas porciones de un contenido consiste en extender el funcionamiento del *ContentName* añadiendo marcadores de segmentación (ver apartado *2. 2. Nombres de Contenido, Marcadores*) y operaciones que permitan moverse por el árbol generado por el *ContentName*. Por ejemplo, un consumidor desea obtener el fichero *"/uc3m/telemática/fichero.pdf"* el cual está dividido en tres segmentos los cuales denominare S1, S2 y S3

1. El consumidor realiza una petición *Interest* al prefijo *"/uc3m/telemática/fichero.pdf"*. En este punto el consumidor desconoce si el contenido está segmentado o no, cuántos segmentos lo componen o si están numerados de forma consecutiva o no.

2. La petición *Interest* alcanza finalmente un nodo que posee una entrada en su *ContentStore* relativa a *"/uc3m/telemática/fichero.pdf"*. A menos que en el campo *ChildSelector* del mensaje Interest especifique lo contrario, se seguirán la reglas del orden canónico por defecto (ver apartado *2. 2. Nombres de Contenido, Orden Canónico*) y se responde con un *ContentObject* cuyo *ContentName* es *"/uc3m/telemática/fichero.pdf/S1"*.

3. Cuando este *ContentObject* retorna al nodo, es decir a la aplicación o el propio módulo CCN solicitante, habrá de generar una nueva solicitud con el *ContentName "/uc3m/telemática/fichero.pdf/S2"*.

4. Finalmente, el proceso se repite para obtener el segmento S3.

Una cuestión a tener en cuenta es que ocurre cuando ya no hay más segmentos por entregar. Con respecto al ejemplo anterior, tras recibir el segmento S3 el nodo solicitante realizaría una nueva petición

"/uc3m/telemática/fichero.pdf/S4" que nunca podría ser satisfecha. Para resolver este problema, la red CCN puede aplicar alguna de las siguientes opciones.

- Transmitir un *ContentObject* del tipo NACK indicando que el contenido *"/uc3m/telemática/fichero.pdf/S4"* no se encuentra actualmente disponible. (Ver apartado *2. 3. 2. Mensaje ContentObject*)

- Añadir en el campo *FinalBlockID* el *ContentName "/uc3m/telemática/fichero.pdf/S3"* de cada uno de los *ContentObject* asociados a S1, S2 y S3, indicando así, que S3 es el último segmento de la secuencia. (Ver apartado *2. 3. 2. Mensaje ContentObject*)

- Previamente, el publicador podría haber registrado los *ContentName* de los 3 segmentos en un *ContentObject* auxiliar cuyo *ContentName* es bien conocido por todos los consumidores. (Ver apartado *2. 5. 1. 2. Funciones relativas a la Capa de Red, Descubrimiento de vecinos y de contenidos*)

De esta forma, se pueden obtener contenidos de los que el solicitante no conoce el nombre completo y exacto.

B) Entrega en orden

Los segmentos TCP pueden ser enviados en orden antes de ser ensamblados en el receptor y entregados a las aplicaciones. CCN no implementa ningún mecanismo de entrega en orden ya que cada par *Interest-ContentObject* es manejado de forma individual. Ello no implica que un conjunto de múltiples paquetes *Interest*; o bien, un conjunto de múltiples de paquetes de datos no puedan ser enviados en bloque sin tener en cuenta una secuencia ordenada.

C) Retransmisión de mensajes

TCP emplea un mecanismo explicito para detectar los segmentos perdidos y retransmitirlos mediante los números de secuencia y de asentimiento.

Por otro lado y al igual que UDP o IP, CCN no describe ningún mecanismo de retransmisión explicito. Sin embargo, se puede establecer los siguientes comportamientos en caso de pérdida de paquetes:

- En caso de haberse perdido un mensaje *Interest*, este será retransmitido con cierta frecuencia por las tablas PIT de los nodos que generaron o reenviaron dicho mensaje *Interest*.

- En caso de haberse perdido un mensaje *ContentObject*, este no será retransmitido explícitamente. Desde el punto de vista del proveedor del contenido, si el solicitante vuelve a enviar un *Interest* para el *ContentObject* perdido, esta solicitud será tratada cómo otra cualquiera.

Finalmente y ya que CCN identifica sus mensajes de forma única en una relación uno a uno, es posible afirmar que CCN se comporta de forma equivalente al mecanismo de asentimientos selectivos de TCP.

D) Duplicación de Paquetes

Los paquetes *Interest* y *ContentObject* podrían duplicarse debido a la distribución multipunto de CCN o retransmisiones en caso de retraso o pérdida de paquetes . Mediante la búsqueda y registro de dichos paquetes en las tablas PIT y *ContentStore*, se puede afirmar:

- En el caso de un Interest duplicado y sus implicaciones en la tabla PIT, el *ContentName* y la *Face* de entrada ya estarían registrados y simplemente se descarta el Interest. En general, el campo *Nonce* permite detectar los duplicados y bucles.

- En el caso de un *ContentObject* duplicado, el *ContentStore* simplemente renueva la entrada asociada en el *ContentStore*. Por lo que realmente, un *ContentObject* duplicado no es un problema.

E) Supresión de bucles en los mensajes

En el caso de los paquetes de datos, realmente este problema no existe, ya que la entrada de la tabla PIT se borra en la primera vuelta del bucle.

Por otro lado y para evitar los bucles de los mensajes *Interest*, estos incorporan el campo *Nonce* cuyo contenido es aleatorio al crear el paquete. De tal forma que cuando un nodo recibe un mensaje *Interest* con un valor *Nonce* ya registrado para ese *ContentName*, simplemente lo descarta.

F) Control de flujo y de congestión

Uno de los factores que mejor han contribuido a la extensión de TCP cómo protocolo de transporte, es su eficiente uso de los recursos disponibles evitando el colapso de los *buffers* de memoria del receptor (control de flujo) y los nodos intermedios (control de congestión).

En comparación con TCP, CCN es extremadamente simple en este aspecto.

- El control de flujo se realiza salto a salto, ya que un *Interest* consigue cómo máximo un paquete de datos [11]. Adicionalmente, un nodo CCN puede crear mecanismos adaptativos por prefijo y por *Face* para conseguir la máxima eficiencia de entrega *Interest-ContentObject*.

- El control de congestión no tiene sentido ya que CCN carece de una perspectiva de la secuencia de nodos por los que pasa la información.

2. 5. 1. 2. Funciones relativas a la Capa de Red

Al igual que en el apartado *2. 5. 1. Capa de Estrategia*, se realizará una comparativa entre el protocolo de red IP y CCN.

A) Descubrimiento de vecinos y contenidos

La definición de vecino en una red IP describe aquel nodo o conjunto de nodos que son accesibles a un salto de distancia y que son capaces de intercambiar e interpretar un determinado tipo de información. Para ello, existen multitud de protocolos y mecanismos auxiliares a nivel de capa 2 o 3, tales cómo ARP, NDP *(Neighbor Discovery Protocol)*, paquetes Hello de OSPF... que permiten identificar los nodos vecinos.

En el caso de CCN, no se implementa un mecanismo de descubrimiento de vecinos, ya que rompería completamente el principio abstracción entre contenido y ubicación del contenido. Sin embargo, esto no implica que cualquier nodo CCN sea aceptable en el dominio, tal cómo se describe en el apartado *2. 5. 2. Capa de Seguridad*, tanto los consumidores cómo los publicadores de contenido han de pasar un proceso de autenticación.

Por otro lado, cuando un nuevo contenido es generado y se le asocia un *ContentName*, es necesario informar al resto de nodos CCN de que ese contenido existe y puede ser solicitado con dicho *ContentName*. Para ello, CCN define una entidad denominada agente de anunciación (Announcement Agent) [11] que registra los nuevos prefijos en un espacio de nombres independiente, por ejemplo, *"/local/CCN/RegistredPrefix"* y los anuncia al resto de nodos CCN. El funcionamiento de este mecanismo es determinado por las reglas de estrategia, por ejemplo, generando un regla cómo "Si el Interest solicita un prefijo no registrado, enviar por la Face por defecto" (Ver apartado *2. 5 .1. 3. Reglas de estrategia y protocolos auxiliares*).

B) Reenvío de paquetes

En un nodo IP, la tabla de reenvío es creada a partir de la información de la tabla de rutas y establece una única interfaz de salida y/o dirección IP del siguiente salto para un determinado prefijo IP aplicando el criterio *Longest Prefix Match*.

En este aspecto, la tabla de reenvío de CCN (tabla FIB) es capaz de utilizar múltiples *Faces* cómo posibles salidas. La utilización de una u otra *Face* de salida es determinado por la reglas de estrategia (Ver apartado *2. 5 .1. 3. Reglas de estrategia y protocolos auxiliares*).

C) Enrutamiento

En las redes IP, un protocolo de enrutamiento permite averiguar la mejor ruta hacia un destino modificando las tablas de reenvío de los nodos involucrados. Existen dos grandes familias de protocolos de enrutamiento [23].

- Protocolos vector distancia, tales cómo RIP, RIPv2 o IGRP, que limitan la perspectiva de la red a un nodo y sus vecinos.
- Protocolos de estado del enlace, tales cómo OSPF o IS-IS, que permiten a los nodos tener una perspectiva completa de la red.

Debido al hecho de que CCN emplea un esquema de reenvío menos restrictivo que el de IP y que la semántica de los *ContentNames* es jerárquica y extensible, CCN es capaz de utilizar una infraestructura de enrutamiento IP existente. En cuanto al tipo de protocolo de enrutamiento que mejor se ajusta a las características de CCN, los protocolos de estado enlace tales cómo OSPF o ISIS son la mejor opción, ya que su estructura TLV (Type-Label-Value) permiten definir entradas en las tablas de rutas específicas para CCN.

2. 5 .1. 3. Reglas de estrategia y protocolos auxiliares

CCN define una regla de estrategia cómo el conjunto de directrices asociadas a una entrada de la tabla FIB y que determinan qué Faces son aplicables y de qué forma serán utilizadas. En una implementación práctica, una regla de estrategia sería la definición de un programa compuesto por instrucciones, disparadores y atributos sobre la *Faces*. Por ejemplo, una regla cómo “Enviar todos los *Interest* para el prefijo “/uc3m.com” por las Faces 1, 2 y 3” no sería posible si no se definiera un programa que tuviera funciones cómo SendtoAll() o SendToBest(), interrupciones del tipo InterestSatisfied o FaceDown y atributos para las Faces tales cómo BroadcastCapabable o MaxPayloadSize.

Adicionalmente, el proyecto CCNx define una serie protocolos para gestionar las Faces disponibles en el nodo, las entradas de la tabla FIB y las reglas de estrategia que se aplicarán a cada entrada de la tabla FIB. Dichos protocolos emplean un mecanismo de petición y respuesta usando mensajes Interest y ContentObject dentro de propio nodo. [24]

Finalmente, en los apartado *4. Convergencia CCN-WSN y 6. Prototipo de sistema CCN para Waspmotes*, describiremos las reglas estratégicas escritas para este proyecto y cómo son implementadas en el entorno de programación de Waspmote, respectivamente. Aquí describiremos los protocolos de gestión de Faces en genérico:

A) Protocolo de Gestión de Faces

Provee las funciones y atributos necesarios para el manejo de las distintas *Faces*. Este protocolo de gestión define una *Face* del nodo cómo un objeto *FaceIntance* que es manejado cómo *ContentObject.* El nodo CCN utiliza este *ContentObject* tanto en la solicitud cómo en la respuesta.

- La solicitud es un mensaje *Interest* cuyo último componente del *ContentName* es el resultado del algoritmo SHA256 sobre *FaceInstance*. Un ejemplo de solicitud sería *"/ccnx/CCNDID/newface/NFBLOB"*, donde CCNDID es el identificador del proceso CCN local y NFBLOB es el *ContentObject* firmado.
- La respuesta es el *ContentObject* que representa el objeto *FaceInstance*.

A continuación describiremos los atributos de un objeto *FaceInstance*:

- *Action.* En la solicitud de una FaceInstance, la acción debe ser indicada, pero no es necesario en la respuesta. Hay tres acciones posibles en este protocolo:
  - "newface". Si la *Face* existe previamente, se genera una respuesta cómo otra *FaceIntance* con toda la información que la define.
  - "destroyface". La solicitud ha de contener el campo *FaceID* y si es posible, eliminará la *Face* del nodo.
  - "queryface". La solicitud ha de contener el campo *FaceID*.
- *PublisherPublicKeyDigest*. Valor SHA256 aplicado sobre la clave publica CCNDID que siempre ha de estar presente en la respuesta.
- *FaceID*. Valor numérico que siempre ha de ser especificado en las respuestas y solicitudes, excepto en la solicitud de la acción "newface".
- *Host*. Valor numérico de la dirección IPv4 o IPv6 remota.
- *IPProto*. Generalmente, contiene la valor correspondiente al protocolo de transporte: TCP valor 6 y UDP valor 17.
- *Port*. Valor numérico entre 1 y 65535 que identifica el puerto de destino o bien el grupo multicast.
- *MulticastInterface*. Si dirección remota contenida en el campo Host es una dirección multicast y el nodo cuenta con múltiples interfaces de salida, MulticastInterface identifica la dirección IP unicast que escuchará la dirección multicast contenida en Host.
- *MulticastTTL*. Valor TTL IP para el trafico multicast. El valor por defecto es 1.
- *FreshnessSeconds*. En el caso de una respuesta, representa el tiempo que permanecerá activo el *Face*. Por otro lado, si es una solicitud, es opcional y será manejado cómo un valor recomendado para el campo *FreshnessSeconds* de la respuesta.

B) Protocolo de Registro de prefijos

Este protocolo emplea el objeto *ForwardingEntry* para manejar las entradas de la tabla FIB. Los atributos definidos para este objeto son:

- *Action.* En la solicitud de una *ForwardingEntry*, la acción debe ser indicada, pero no es necesario en la respuesta. Hay tres acciones posibles en este protocolo:
  - "prefixreg". Para registrar o renovar un prefijo en un determinada *Face*.
  - "selfreg". Para registrar o renovar un prefijo en la *Face* actual.
  - "unreg". Para borrar un prefijo de un *Face*.
- *PublisherPublicKeyDigest*. Valor SHA256 aplicado sobre la clave publica CCNDID que siempre ha de estar presente en la respuesta.
- *FaceID*. Valor numérico que siempre ha de ser especificado en las respuestas y solicitudes, excepto en la solicitud de la acción "selfreg".
- *Name*. Prefijo a registrar o a borrar en una *Face*.
- ForwardingFlags. Consiste en 8 flags de un bit cada uno. De menor a mayor peso, los flags definidos son:
  - 1º Bit. CCN_FORW_ACTIVE. Indica que la entrada está activa y que los mensajes *Interest* pueden reenviados siguiendo dicha entrada.
  - 2º Bit. CCN_FORW_CHILD_INHERIT. Indica que la entrada podría ser utilizada incluso si existiera una entrada preferente según el criterio *Longest Prefix Match*.
  - 3º Bit. CCN_FORW_ADVERTISE. Indica que el prefijo podría ser anunciado a otros nodos.
  - 4º Bit. CCN_FORW_LAST. Indica que la entrada debería ser usada en último lugar cuando ninguna otra funciona. El uso de este flag impide que la *Face* pueda recibir *Interest* de carácter local.
  - 5º Bit. CCN_FORW_CAPTURE. Se utiliza conjuntamente con los flags CCN_FORW_CAPTURE_OK y CCN_FORW_CAPTURE_OK, indicando que un prefijo no más corto que el nombre especificado en el objeto podría ser usado priorizando el bit CCN_FORW_CHILD_INHERIT. Para ello, los flags CCN_FORW_CAPTURE_OK y CCN_FORW_CHILD_INHERIT han de estar activos en la entrada.
  - 6º Bit. CCN_FORW_LOCAL. Limita el alcance de la entrada a aplicaciones dentro del nodo, de forma equivalente al valor 1 en campo *Scope* del mensaje *Interest*.
  - 7º Bit. CCN_FORW_TAP. Indica que la entrada sea usada inmediatamente.
  - 8º Bit. CCN_FORW_CAPTURE_OK. Determina si CCN_FORW_CAPTURE tiene efecto o no sobre la entrada.

- *FreshnessSeconds*. En el caso de una respuesta, representa el tiempo que permanecerá activo la Face. Por otro lado, si es una solicitud, es opcional y será manejado cómo un valor recomendado para el campo FreshnessSeconds de la respuesta.

C) Protocolo de Selección de Estrategias

El objetivo de este protocolo que asociar una regla de estrategia concreta con una entrada de la FIB. El hecho de que este protocolo tenga la capacidad de decidir qué regla se aplicará lo hace especialmente útil para los protocolos de enrutamiento.

Para realizar estas tareas de selección de estrategias, se define un objeto *StrategySelection* cuyos atributos son:

- *Action.* En la solicitud de una *StrategySelection*, la acción debe ser indicada, pero no es necesario en la respuesta. Hay tres acciones posibles en este protocolo:
  - "setstrategy". Asocia una estrategia con un prefijo.
  - "getstrategy". Obtiene la estrategia asociada a un prefijo.
  - "removestrategy". Elimina la estrategia asociada a un prefijo.
- *PublisherPublicKeyDigest*. Valor SHA256 aplicado sobre la clave publica CCNDID que siempre ha de estar presente en la respuesta.
- *StrategyID*. Limitado a 15 caracteres, ha de estar presente en todas las respuestas y en las solicitudes que contengan la acción "setstrategy".
- *StrategyParameters*. Contiene los parámetros necesarios para la ejecución de la regla de estrategia. Se trata de una cadena de texto cuyo formato puede variar de una regla a otra, sin embargo, se recomienda seguir un formato URI.
- *FreshnessSeconds*. En el caso de una respuesta, representa el tiempo que permanecerá activa la *Face*. Por otro lado, si es una solicitud, es opcional y será manejado cómo un valor recomendado para el campo *FreshnessSeconds* de la respuesta.

En caso de una solicitud infructuosa, el protocolo puede no dar una respuesta o responder con un *ContentObject* del tipo NACK cuyo contenido sea un mensaje describiendo el problema.

2. 5. 2. Capa de Seguridad

El proyecto CCNx presta una especial atención a las cuestiones criptográficas de los contenidos. Esto se debe al hecho de que CCN no puede aplicar mecanismos de seguridad directamente sobre los nodos o sobre medio por el que viajan los mensajes. Por lo que toda la carga de seguridad recae sobre los mensajes CCN, de tal forma, que los mensajes son verificados mediante firma digitales, los contenidos en sí mismos son protegidos mediante encriptación y los *ContentName* son certificados mediante la adhesión de un componente criptográfico. En general CCNx, definen tres grupos de algoritmos [25].

- Algoritmos de la firma digital. *(Signature Algorithms)*
- Algoritmos de encriptación.(*Encription Algorithms*)
- Algoritmos de componente implícita.(*Built-in Digest Algorithm)*

Por otro lado, en el apartado *2. 5. 2. 4. Gestión de claves públicas,* se describirá cómo obtienen y usan los nodos CCN las claves públicas necesarias para el uso y verificación de la información cifrada por los algoritmos criptográficos.

2. 5. 2. 1. Algoritmos de la firma digital

En el apartado *2. 3. 2. Mensaje ContentObject, Firma Digital,* se describió la estructura interna de la firma digital destinada a verificar la autenticidad de un *ContentObject*. La elección del algoritmo criptográfico y la generación de las firmas digitales es competencia del publicador del contenido.

El criterio de selección entre uno u otro algoritmo depende las necesidades del tipo de contenido, así cómo, de las restricciones computacionales de los nodos y de la propia red. Independiente del algoritmo y la clave escogidos, un publicador CCN tiene dos formas de asignar una firma digital a un *ContentObject* bien de forma individual o bien de forma agregada.

A) Firma individual del *ContentObject*

Aplicando esta solución, la firma digital sería generada a partir de los *ContentName*, *SignedIfno* y el contenido empleando librerías criptográficas estándar e indicado el algoritmo usado en el campo *DigestAlgorithm*. El resultado sería almacenado en el campo *SignatureBits* y el campo *Witness* sería omitido.

B) Firma agregada de los *ContentObject*

Se genera una firma agregada para varios *ContentObject* añadiendo en el campo *Witness* la información necesaria para la verificación individual de cada bloque de datos.

La utilización de una u otra asignación de firmas depende en gran medida de la cantidad de claves públicas disponibles y de los costes computacionales de su generación y verificación. Por ejemplo, si el publicador firmara cada *ContentObject* de forma individual, el publicador asumiría mayores costes de computación mientras que el tamaño de las firmas sería menor. Por otro lado, si las firmas son generadas de forma agregada, los costes de verificación serían menores, sin embargo habría que incluir la información oportuna en el campo *Witness* por lo que el tamaño de la firma sería mayor.

2. 5. 2. 2. Algoritmos de encriptación

Este tipo de algoritmos son utilizados para la protección del contenido en sí mismo dentro del *ContentObject* entre el publicador y el consumidor. El algoritmo escogido ha de ser el mismo para el publicador y el consumidor mientras que su utilización ha de ser completamente opaca al normal funcionamiento de CCN.

2. 5. 2. 3. Algoritmos de componente implícita

CCNx define tres localizaciones y usos dentro de los mensajes CCN (ver apartados *2. 3. 1. Mensaje Interest* y *2. 3. 2. Mensaje ContentObject*). En todos los casos el algoritmo de cifrado utilizado es SHA256, aunque actualmente el proyecto CCNx está desarrollando la implementación del algoritmo SHA3. Ubicaciones y usos:

- En los campos *PublisherPublicKeyDigests* de los mensajes *Interest* y *ContentObject*, sus valores cifrados SHA256 son generados a partir de las claves públicas obtenidas del publicador. Dichos valores cifrados han de coincidir en cada par *Interest-ContentObject* para asegurar la asociación inequívoca entre ellos.

- Componente implícito SHA256. cómo ya se explicó en el apartado *2. 2. Nombres de Contenido* se trata de un componente añadido en la última posición del *ContentName* solicitado en el mensaje *Interest*. Dicho componente es generado a partir del *ContentObject* solicitado.

- En el campo *Exclude* del mensaje *Interest*, es posible especificar qué *ContentObjects* quedan excluidos en la solicitud. Este mecanismo de exclusión puede llevarse a cabo indicando el valor resultante del algoritmo SHA256 sobre un *ContentObject* recibido, de tal forma que se evita la recepción de ese *ContentObject* de nuevo.

2. 5. 2. 4. Gestión de claves públicas

En primer lugar, CCN maneja las claves cómo cualquier otro contenido, por lo cual una clave pública es firmada y posee un *ContentName* propio.

Cuando un mensaje *Interest* es enviado al publicador y el *ContentObject* firmado llega al consumidor, este ha de obtener la clave para generar y verificar el valor del campo *PublisherPublicKeyDigest*. Hay dos campos del *ContentObject* encargados de indicar cómo obtener la clave (ver apartado *2. 3. 2. Mensaje ContentObject*).

- *PublisherPublicKeyDigest*. Almacena el resultado del algoritmo SHA256 aplicado sobre la clave pública. El descifrado de esta información puede realizarse rápidamente a partir de otra clave almacenada en la cache local del nodo.

- *KeyLocator*. Puede contener el *ContentName* necesario para obtener la clave pública o la clave pública en sí misma. En el caso de contener un *ContentName*, habría de generarse un nuevo *Interest* y se recibiría un *ContentObject* firmado del tipo KEY con la clave pública.

Ya que las claves distribuidas mediante un *ContentName* son entregadas en un *ContentObject* firmado, una cuestión importante a tener en cuenta es cómo obtener la clave que permita verificar otras claves. Al igual que los *ContentName*, las claves públicas son jerárquicas, es decir, si un consumidor posee la clave pública de una organización de proveedores es capaz de validar las claves de los distintos departamentos y usuarios subyacentes a la organización [11].

3. Redes Inalámbricas de Sensores

Las redes WSN (*Wireless Sensor Networks*) surgieron gracias al abaratamiento y miniaturizaron de elementos computacionales, permitiendo así, la dispersión de nodos autónomos creando redes extensas con pocos recursos. WSN se compone de nodos con capacidades sensoriales que transmiten su información de forma inalámbrica. Este tipo de redes tienen múltiples aplicaciones, tales cómo: control medioambiental, domótica, aplicaciones médicas, seguridad.... Las características de la redes WSN son [7][26]:

- **Topología dinámica y descentralizada.** En el caso de comunicaciones WIFI, esto implica típicamente conexiones ad-hoc entre los nodos.
- **Autonomía.** Cada nodo ha de poseer sus propias funciones de transmisión, enrutamiento y procesamiento de la información.
- **Facilidad despliegue.** La incorporación de nuevos nodos a la red no puede traer consigo configuraciones complejas y particularizadas.
- **Tolerancia a errores.** Al ser un medio inalámbrico y descentralizado, los errores son posibles.
- **Eficiente uso energético.** Su importancia depende en gran medida de dónde se despliega la red, por ejemplo, si una red WSN es desplegada en el interior de una oficina con fácil acceso a un suministro eléctrico, la eficiencia energética no sería factor crítico.
- **Limitadas capacidades de almacenamiento y procesamiento.**

En cuanto a la interfaz de comunicación, las WSN tienden a utilizar una interfaz WiFi o IEEE 802.15.4 aunque también hay otras opciones cómo [7]:

- 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*) es el actual enfoque para conectar WSN a Internet.
- CoAP (*Constrained Application Protocol*) provee de servicios WEB-UDP para WSN.

3. 1. Valoraciones de diseño

Una vez explicados los aspectos básicos de las redes WSN, en los siguientes suba-apartados analizamos las valoraciones de diseño realizadas para este proyecto. Las siguientes valoraciones tan solo abarcan aspectos inalámbricos, sensoriales y de consumo del proyecto. Dichas valoraciones, se tendrán en cuenta en el diseño del módulo CCN-WSN que se analizará en el apartado *4. Convergencia CCN-WSN.*

3. 1. 1. Módulo Inalámbrico Waspmote

La interfaz inalámbrica escogida es un módulo Wifi con una antena de 2 dBi, por las siguientes razones (Ver también Tabla 2 en el apartado *5. 1. 2. Módulo WiFi*).

- **Rango de cobertura.** En la realización de este proyecto se requiere cómo mínimo unos pocos metros de cobertura. Por ello, el rango de 50 a 100 metros que ofrece esta opción es más que suficiente.

- **Potencia de transmisión y recepción.** Este factor tiene dos implicaciones importantes, rango de cobertura y consumo energético. Ambas implicaciones no son criticas para la resolución de este proyecto ya que la distancia entre los nodos es despreciable y los dispositivos cuentan con un suministro eléctrico constante.

- **Facilidad de despliegue y coste .** Emplear conexión WiFi simplifica la arquitectura global de comunicaciones porque es posible acceder a la red WSN desde una amplia variedad de dispositivos. Es además una solución barata y flexible, ya que es posible conectar los sensores a la red a través de cualquier router inalámbrico (o punto de acceso) WiFi de los que hay muchos disponibles en el mercado a bajo coste. También nos encontramos con puntos de acceso WiFi desplegados en muchos escenarios, pudiendo aprovecharse para conectar sensores de nuestra red, lo que simplifica mucho el despliegue de la red de sensores en un entorno en el que ya tenemos una infraestructura de cobertura WiFi.

Por otro lado, los parámetros WiFi escogidos para este proyecto son.

- Modo: Infraestructura.
- Encriptación: Ninguna
- Canal: Cualquiera
- Nº Nodos: 3.
- Distancia máxima entre nodos: Aproximadamente 1 metro.
- Router inalámbrico Asus RT-16 [39].

3. 1. 2. Módulo Sensor

En este proyecto se analiza el paradigma CCN aplicado a las redes de sensores pero nos centramos en la parte de comunicaciones. Por lo tanto, no usamos sensores reales para obtener los datos que se van a transferir. En su lugar, los datos transportados son generados por la propia aplicación (ver apartado *6. 2. 2. Inicialización del programa*).

3. 1. 3. Distribución de los nodos

En una red WSN, no existe ninguna estructura concreta, dependiendo de su ubicación el conjunto de nodos pueden formar diferentes topologías (árbol, malla, anillo...) con diferentes implicaciones.

Una de las implicaciones de diferentes topologías es el retardo extremo a extremo [27] que aumenta con el número de saltos. Por tanto, una topología en malla completa (la cual permite una conexión directa entre todos y cada uno de los nodos) presentara menor retardo medio que una topología en anillo, por ejemplo.

3. 1. 4. Consumo energético

Uno de los inconvenientes de la autonomía de los nodos WSN es que dichos nodos pueden estar ubicados en lugares donde el acceso a la corriente eléctrica o la recarga de la batería son difíciles de conseguir. Por ello, un eficiente uso energético es una de las características en las redes WSN.

En el caso de este proyecto, el consumo energético no ha sido considerado cómo un factor crítico en las cuestiones de diseño e implementación. No obstante, existen ciertas limitaciones del dispositivo Waspmote que dificultan la aplicación de dicho factor y cómo se describirán en el apartado *6. 1. Limitaciones de dispositivo.*

4. Convergencia CCN-WSN

En los apartados 2 y 3, se analizaron por separado el proyecto CCNx y las redes WSN. Finalmente en el presente apartado, se analiza la convergencia CCN-WSN desarrollada en este proyecto.

El planteamiento de nuestra propuesta de solución CCN-WSN es emplear las directrices del CCNx adaptándolas a los requisitos de las redes WSN. En este capítulo describiremos el funcionamiento lógico de la solución, y será en el capítulo 6 donde se explicará la implementación concreta considerando las restricciones hardware y software propias del dispositivo Waspmote.

4. 1. Nombres de Contenido CCN-WSN

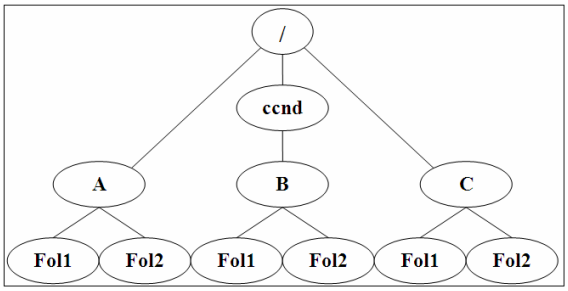
Al que igual que el proyecto CCNx, los *ContentNames* utilizados en nuestro diseño habrán de seguir un formato URI aunque con pequeñas diferencias en la estructura general del *ContentName*.

- **Prefijo globalmente enrutable.** Esta parte consiste un identificador del origen del contenido, accesible por Internet. Debido al hecho de que un nodo WSN no necesita ser accesible fuera de la propia red WSN, los *ContentName* de este proyecto no contienen este tipo de prefijo.

- **Estructura organizativa.** En nuestro caso, la estructura de los componentes del ContentName no representará la ruta un fichero. De hecho, el manejo de ficheros no es imprescindible en la resolución de este proyecto.

Por simplicidad, tan sólo se ha definido un espacio de nombres para todos los contenidos. La siguiente figura ilustra el árbol de componentes.

Figura. 10. Espacio de nombres CCN-WSN



4. 2. Mensajes de CCN-WSN

En el apartado *2. 3 Mensajes CCN* describimos todos los campos de los mensajes *Interest* y *ContentObject* en CCNx, e identificamos los campos que podían ser omitidos en nuestro escenario objetivo.

4. 2. 1. Mensaje *Interest*

Todo mensaje *Interest* ha de contener al menos el prefijo del nombre a solicitar. Adicionalmente, necesitamos los siguientes campos (ver apartado *2. 3. 1. Mensaje Interest*): *Name, ChildSelector, PublisherPublicKeyDigest y InterestLifetime.*

4. 2. 2. Mensaje ContentObject

Como ya se explicó en el apartado *2. 3. 2. Mensaje ContentObject*, este mensaje ha de contener obligatoriamente la firma digital del contenido (*Signature*), el *ContentName* del contenido, ciertos campos de *SignedInfo* cómo *PublisherPublicKeyDigest*. o *Timestamp* y el propio contenido a transportar. En nuestro diseño se incluyen los campos opcionales *Type* y *FreshnessSeconds*.

4. 2. 3. Mecanismos auxiliares

Los mecanismos auxiliares implementados en el módulo CCN-WSN serán:

- **Marca de antigüedad (Staleness).** Definida por un único bit en cada entrada del *ContentStore* está diseñado para activarse pasado el tiempo marcado por el campo *FreshnessSeconds* a partir del valor *Timestamp* del mensaje *ContentObject* (también denominado *CreationTimestamp* en la tabla *ContentStore*). La función de esta marca es priorizar el descarte de las entradas de la tabla *ContentStore* cuando la diferencia los valores *CreationTimestamp* y *LastTimeUsed* superan cierto intervalo de tiempo. No obstante, en los apartados *2. 3. 1. Mensaje Interest* y *2. 3. 3. Mecanismos auxiliares*, se describió la posibilidad de entregar contenidos anticuados indicando el valor 0x4 en el campo *AnswerOringinKind* del mensaje Interest. Dado que este campo no es implementado en nuestro diseño, se aplicará la opción de CCN por defecto en este aspecto, es decir, las entradas del *ContentStore* con la marca de antigüedad activa no serán entregadas en las peticiones Interest recibidas.

- **Marcas de tiempo (*Timestamp*).** Necesarias para establecer el correcto funcionamiento del mensaje ContentObject a través del campo *Timestamp y* de las tablas *ContentStore* y *PIT* a través de los campos *CreationTimestamp*, *LastTimeUsed* y *LastTimeTried*, respectivamente. En las primeras etapas del diseño se optó por emplear marcas de tiempo de 32 bits en unidades de segundos. Sin embargo, debido una serie de limitaciones del dispositivo que describiremos en el apartado 6, las marcas de tiempo no han sido representadas mediante valores numéricos sino por cadenas de caracteres.

4. 3. Estructura de un nodo CCN-WSN

En el presente apartado, se definen la estructura de la tablas y Faces de un nodo.

4. 3. 1. Content Store *(buffer memory)*

El formato del *ContentStore* ha de corresponderse con los campos definidos en el *ContentObject* tales cómo *PublisherPublicKeyDigest*, *Timestamp*, *Type* o *FreshnessSeconds* y otros necesarios para la operación normal del *ContentStore* cómo el bit de antigüedad y el campo *LastTimeUsed*. Por otro lado, la gestión de memoria escogida para este diseño es LRU (*Least Recently Used)* que obliga a almacenar la marca de tiempo de la última ocasión que un contenido fue solicitado en el campo *LastTimeUsed*. En conclusión, el siguiente ejemplo refleja el formato de las entradas del ContentStore.

Figura. 11. Entrada en la tabla ContentStore CCN-WSN

ContentName	PublisherPublicKeyDigest	Type	Staleness Bit	Creation Timestamp	FreshnessSeconds	LastTimeUsed
/carp1/carp2/fich.txt	0x5642ADFC	DATA	0	13:35:41	10	13:35:44

4. 3. 2. PIT (Pending Interest Table)

En este caso, la tabla PIT ha de ser consecuente con el formato del mensaje *Interest* (ver apartado *4. 2. 1. Mensaje Interest*). Para evitar la sobre-escritura de las entradas de la tabla PIT, tal y cómo explicó en el apartado *2. 4. 2. PIT (Pending Interest Table)*, cada entrada en la tabla PIT representara un mensaje *Interest* recibido. La figura muestra el formato de las entradas en la tabla PIT.

Figura. 12. Entrada en la tabla PIT CCN-WSN

RequestedPrefix	Creation Timestamp	LastTimeTried	Tries	InterestLifeTime	RequestedFace
/carp1/carp2/fich.txt/	18:19:04	18:19:06	2	10	1

4. 3. 3. FIB (Forwarding Information Base)

A priori, la tabla FIB puede parecer sencilla en comparación el resto de tablas de nuestro diseño, ya que únicamente asocia un prefijo con una lista de *Faces* de salida. Sin embargo, la utilización de dichas entradas depende de las reglas de estrategia definidas que podrían ser tan simples cómo, por ejemplo, "Reenvía el *Interest* por todas la Faces de salida" o tan complejas cómo, por ejemplo, "Si determinados componentes aparecen en el *Interest*, prioriza el reenvío por el *Face* 1, sino es resuelto en un cierto tiempo, reenvía el *Interest* de forma secuencial por el resto de Faces decrementando el *InterestLifetime*". En el apartado *4. 4. 1. Capa de Estrategia en CCN-WSN*, se definirán las reglas de estrategia para este diseño.

Independientemente de las reglas de estrategia establecidas, podemos definir el siguiente formato de las entradas de la tabla FIB.

Figura. 13. Entrada en la tabla FIB CCN-WSN

Prefix	StrategyRuleID	OutputFaces
/carp1/carp2	Rule1	1, 5, 6

4. 3. 4. Procesamientos de mensajes CCN-WSN

En el apartado *2. 4. 4. Procesamientos de mensajes*, se describió el procesamiento estándar del mensaje *Interest* y *ContentObject*, respectivamente. A continuación, describiremos el procesamiento de los mensajes en el diseño final de este proyecto.

A) Procesamiento del mensaje *Interest*

El primer a paso a la llegada de mensaje *Interest* es verificar su autenticidad. Para ello, hay que realizar dos procesos de verificación.

- Generar un componente SHA256 y compararlo con el último componente del *ContentName* del mensaje *Interest* asegurando así que el mensaje no ha sufrido errores o manipulaciones durante la transmisión.
- Comprobar que el valor contenido en el campo *PublisherPublicKeyDigest* se corresponde con el valor SHA256 de la clave publica para ese contenido en la entrada del *ContentStore* legitimando el derecho del consumidor a obtenerlo.

Si el mensaje *Interest* no pasa ambos procesos de verificación, simplemente se descarta el mensaje. En caso contrario, si el mensaje *Interest* es legítimo, el proceso continúa en la siguiente secuencia de pasos.

1. *ContentStore*. Dependiendo de las entradas contenidas en *ContentStore* podemos distinguir tres posibles casos asumiendo que todas las entradas en el *ContentStore* tienen su bit de antigüedad desactivado (ver apartado *2. 3. 3. Mecanismos auxiliares, Marca de Antigüedad*).

- Si hay una coincidencia en el *ContentStore* con el *ContentName* solicitado, se envía el *ContentObject* correspondiente por la *Face* de entrada del mensaje *Interest*.
- Si hay dos o más coincidencias en el *ContentStore* con el *ContentName* solicitado, se aplicará el valor del campo *ChildSelector*. Por ejemplo, si el prefijo solicitado es *"/uc3m/Carpeta1"* y *ContentStore* tiene 3 entradas con ese prefijo *"/uc3m/Carpeta1/A.txt"*, *"/uc3m/Carpeta1/B.txt"* y *"/uc3m/Carpeta1/C.txt"*, se puede afirmar que:
  - Cuando ChildSelector es igual a 0 (hijo más a la izquierda), se envía el *ContentObject "/uc3m/Carpeta1/A.txt"* por la *Face* entrada del mensaje *Interest.*
  - Cuando ChildSelector es igual a 1 (hijo más a la derecha), se envía el *ContentObject "/uc3m/Carpeta1/C.txt"* por la *Face* entrada del mensaje *Interest.*
- Finalmente, si no hay ninguna coincidencia, el proceso continúa en la tabla PIT.

2. PIT. Tal y cómo indicó en el apartado *4. 3. 2. PIT (Pending Interest Table*), el procesamiento de los mensajes *Interest* en la tabla PIT se desvía del procedimiento estándar del proyecto CCNx indicado en el apartado a *2. 4. 2. PIT (Pending Interest Table).* Por ello, describiremos cómo se procesa los mensajes *Interest* en la tabla PIT en nuestro diseño.

2.1. Se busca el prefijo solicitado en el mensaje *Interest* en la tabla.

- Si hay una coincidencia con el prefijo, procede comprobar si el campo *RequestedFace* ya está registrado.
  - En caso de que *RequestedFace* esté registrado, ello implica que el mensaje *Interest* ya fue recibió por esa *Face* y simplemente se descarta el *Interest*.
  - En caso de que *RequestedFace* no esté registrado, implica que el mensaje *Interest* no fue recibió por esa *Face* y, por tanto, ha de crearse una nueva entrada indicando *RequestedPrefix, CreationTimestamp, InterestLifeTime y RequestedFace.* Tras registrar la entrada el proceso continua en la tabla FIB.

- Si hay no coincidencia con el prefijo, se genera una nueva entrada indicando *RequestedPrefix, CreationTimestamp, InterestLifeTime y RequestedFace.* Tras registrar la entrada el proceso continua en la tabla FIB.

2.2. Finalmente y de forma periódica, los mensajes *Interest* pendientes son reenviados por las *Faces* por la tabla FIB. A su vez, el tiempo de vida de la entrada en la PIT depende de los valores contenidos en *InterestLifeTime*. *Los valores Tries y LastTimeTried* actúan cómo registros de estos envíos periódicos y no son determinantes a la hora de modificar el tratamiento de las entradas de la PIT.

3. FIB. Mantiene en gran medida el procedimiento estándar de CCNx busca el prefijo solicitado en la tabla según el criterio *Longest Prefix Match*, aplica la regla de estrategia contenida en *StrategyRuleID* sobre las Faces de salida indicadas *OutputFaces*. No obstante, en el apartado *4. 5. 1. Capa de Estrategia en CCN-WSN*. describiremos las reglas de estrategia que ayudarán a comprender mejor el funcionamiento de esta tabla.

B) Procesamiento del paquete *ContentObject*

Al igual que el mensaje *Interest*, un mensaje *ContentObject* ha de pasar dos procesos de autenticación.

- Firma digital. Al recibir el *ContentObject*, se genera una nueva firma a partir de los campos Name, *SignedInfo* y *ContentName* aplicando el algoritmo indicado en el campo *DigestAlgorithm* y comparando el resultado con el valor contenido en el campo *SignatureBits* del *ContentObject* recibido. De esta forma, se verifica no solo la identidad del publicador de contenido sino también la integridad del mensaje. En el apartado *4. 5. 2. 1. Algoritmos de la firma digital*, se describirán los detalles de las firmas digitales empleadas en este proyecto.

- *PublisherPublicKeyDigest*. Al igual que en el campo *PublisherPublicKeyDigest* del mensaje *Interest*, este campo contiene el valor SHA256 de la clave pública para ese contenido de tal forma que el nodo pueda validar el contenido.

Una vez se ha verificado el *ContentObject* recibido, pasaremos a explicar cómo lo manejan las tablas *ContentStore* y PIT.

1. ContentStore. Si hay coincidencia exacta con el *ContentName*, se sobrescribe el *ContentObject* de la tabla renovando los valores *StalenessBit*, *CreationTimestamp*, *FreshnessSeconds* y *LastTimeUsed*. Sino hay coincidencia, se añade una entrada en el *ContentStore*. En cualquiera de los dos casos, el proceso continúa en la tabla PIT.

2. PIT. A la llegada del *ContentObject*, se busca su *ContentName* entre los prefijos registrados en la tabla PIT. Si hay una o más entradas que pueden ser satisfechas con el *ContentObject* en cuestión, se genera una copia para cada una de las entradas y se envían por el *Face* indicada en el campo *RequestedFace*. Una vez enviados los *ContentObjects*, las entradas de la PIT satisfechas son eliminadas. Por supuesto si no hay ninguna coincidencia, no se envía ningún *ContentObject*.

4. 4. Modelo de Protocolo en CCN-WSN

En este proyecto hay tres posibles localizaciones del módulo CCN-WSN en la pila TCP/IP: situarlo sobre la capa transporte, la capa de Internet o la capa de acceso a red.

Por simplicidad a la hora de implementar las funciones de envío y recepción de Waspmote, el módulo de CCN-WSN se posicionará sobre la capa de transporte.

En cuanto al protocolo transporte escogido, se ha optado por UDP no solo porque al igual que CCN es un protocolo de entrega no fiable, sino también por la naturaleza móvil y descentralizada de las redes WSN que podrían alterar la normal funcionamiento de una conexión TCP. Por otro lado, el protocolo de red escogido es IP versión 4, debido principalmente que las librerías estándar de Libelium solo permiten este protocolo de red.

4. 4. 1. Capa de Estrategia en CCN-WSN

Como ya se explico en el apartado *2. 5. 1. Capa de Estrategia*, la capa de Estrategia engloba todos aspectos relativos al transporte de la información sintetizados en forma de reglas de estrategia. Por consiguiente, describiremos únicamente los aspectos específicos de la implementación CCN-WSN.

A) Secuenciación de mensajes

En este diseño, no es necesario ningún mecanismo de secuenciación, ya que la carga de datos se limita a unos pocos bytes en todos los *ContentObject* independientemente del nombre de contenido.

B) Descubrimiento de vecinos y contenidos

Dado que el número de nodos y el espacio de nombre son mínimos en esta implementación, no ha sido necesario crear mecanismos que permitan a los nodos averiguar qué nodos o contenidos están disponibles.

D) Enrutamiento

No se ha implementado ningún protocolo enrutamiento o mecanismo auxiliar que asocie un prefijo a una *Face* de salida concreto, en este proyecto usamos rutas estáticas y se deja como futura línea de trabajo el interesante reto de los algoritmos de encaminamiento adecuados en WSN.

E) Reglas de estrategia y protocolos auxiliares

Tal y cómo describimos en el apartado *2. 5 .1. 3. Reglas de estrategia y protocolos auxiliares*, el proyecto CCNx define una regla de estrategia cómo un programa encargado de asociar prefijos de contenido a un *Face* específica, y una serie protocolos de petición-respuesta encargados de gestionar las *Faces*, las entradas de la tabla FIB y las reglas de estrategia que se aplicarán.

La única regla de estrategia definida en este proyecto es extremadamente simplemente.

"Aplicando Longest Prefix Match, todo mensaje Interest será enviado por los valores OutputFace de la tabla FIB"

Dado que nuestra implementación se reduce una única regla de estrategia, y tanto las Faces cómo las entradas de la FIB son estáticas, no se ha implementado ningún protocolo de gestión. Sin embargo, tanto reglas de estrategia más complejas como protocolos de gestión pueden ser trabajos futuros.

4. 4. 2. Capa de Seguridad CCN-WSN

Aunque CCN emplea varios mecanismos para asegurar la integridad de los contenidos y la confidencialidad de los consumidores y publicadores, en este proyecto las políticas son un tanto laxas, sin embargo, emplearemos la misma estructura del apartado *2. 5. 2. Capa de Seguridad* para explicarlas.

4. 4. 2. 1. Algoritmos de la firma digital para CCN-WSN

Como ya se explicó en el apartado *2. 3. 2. Mensaje ContentObject*, el campo *Signature* es un campo obligatorio para todo *ContentObject* que a su vez se compone de tres subcampos.

- *DigestAlgorithm.* En todos los casos será SHA256, por lo que el valor contenido en este campo no será verificado.
- *Witness*. cómo se ha optado por firmar cada *ContentObject* de forma individual, no necesario comprobar este campo.
- *SignatureBits*. Contiene el resultado final de la aplicación del algoritmo sobre el *ContentObject*.

4. 4. 2. 2. Algoritmos de encriptación

Para no cargar a los dispositivos con tareas de encriptación de contenidos, no se ha implementado ningún algoritmo de encriptado.

4. 4. 2. 3. Algoritmos de componente implícita

Al igual que en el caso de la firmas digitales, el algoritmo utilizado es SHA256 y aplicado en dos situaciones.

- En los campos *PublisherPublicKeyDigests* de los mensajes *Interest* y *ContentObject*, generando el resultado a partir de una clave publica.
- En la última posición del *ContentName* del mensaje *Interest*. generando el resultado a partir del propio *ContentName*.

4. 4. 2. 4. Gestión de claves públicas

Por simplicidad habrá una única clave pública para todos los nodos y para todos los contenidos. Esta clave pública será almacenada localmente en cada nodo por lo que no es necesario el uso del campo *KeyLocator* o un espacio de nombres especial para la distribución de claves

5. Plataforma WASPMOTES

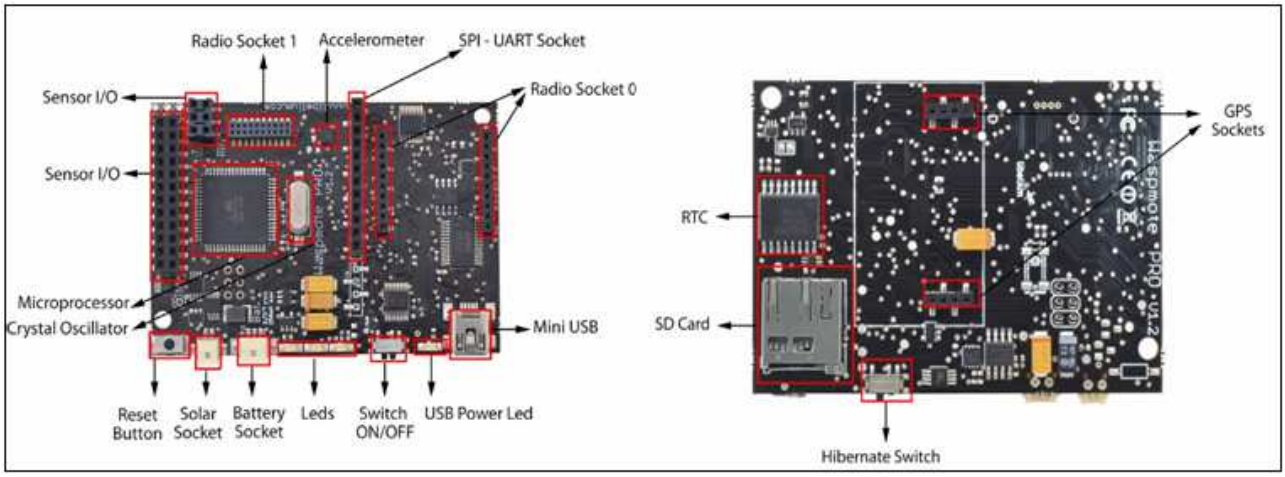
En el presente apartado, se describe los aspectos hardware de los nodo Waspmote empleando la información suministrada por la compañía Libelium [28][29][30], así cómo, los aspectos software.

5. 1. Aspectos Hardware

Cada uno de los nodos Waspmote empleados durante la realización del proyecto constan de los siguientes elementos que se explicarán con más detalle en apartados posteriores:

- Placa madre y batería (ver apartado *5. 1. 1. Placa madre y elementos auxiliares*)
- Módulo Wifi. (ver apartado 5. *1. 2. Módulo Wifi*).
- Ningún módulo sensor.

Figura. 14. Vista anterior y posterior de un Waspmote[30]



Existen otras plataformas con las que construir una red WSN, tales cómo, TinyOS, Hoperf, Accsense, MAXFOR Tecnology, Coalesenses...[31], pero no presentan una modularidad y versatilidad similar a Libelium.

5. 1. 1. Placa madre y elementos auxiliares

Actualmente, existe dos versiones del dispositivo Waspmote. Ambos versiones poseen características hardware muy similares pero ciertas funciones de las librerías que difieren una versión a otra. La siguiente tabla muestra las especificaciones del dispositivo Waspmote V1.1. y Waspmote Pro V1.2 [28][32].

Tabla 1. Hoja de características de Waspmote V1.1. y Waspmote Pro V1.2

		Waspmote V1.1	**Waspmote PRO**
Datos Generales	**Microcontrolador**	ATmega 1281	ATmega 1281
	Frecuencia	8 MHz	14.7456 MHz
	SRAM	8 kB	8 kB
	EEPROM	4 kB	4 kB (1kB reservado)
	FLASH	128 kB	128 kB
	SD Card	2 GB	2 GB
	Peso	20 g	20 g
	Dimensiones	73.5 x 51 x 13 mm	73.5 x 51 x 13 mm
	Rango temperatura	-20oC + 65oC	-10oC + 65oC
	Relog	RTC (32 kHz)	RTC (32 kHz)
Consumo	**ON**	9 mA	15 mA
	Suspensión	62 uA	55 uA
	Suspensión	62 uA	55 uA
	Hibernación	0.7 uA	0.06 uA
Entradas/Salidas		7 Analog Inputs 8 Digital I/O 2 UART 1 I2C 1 USB	7 Analog Inputs 8 Digital I/O 2 UART 1 I2C 1 SPI 1 USB
Datos Eléctricos	**Voltaje batería**	3.3V – 4.2V	3.3V – 4.2V
	Carga USB	5V – 100 mA	5V – 100 mA
	Carga panel solar	6 – 12 V – 280 mA	6 – 12 V – 280 mA
	Botón auxiliar (RTC)	3V	No necesario

5. 1. 1. 1. Elementos de Almacenamiento

Tal cómo observarse en la Tabla 1, el dispositivo Wapmote cuenta con 4 espacios de almacenamiento cada uno con un propósito.

- **Memoria Flash (128KB).** En ella, se almacena el programa escrito para una implementación concreta y un pequeño programa bootloader encargado del arranque del dispositivo. Dado el tamaño de esta memoria y su composición, hay que evitar borrar el programa bootloader o cargar programas demasiado extensos.

- **Memoria EEPROM (4KB).** En ella, se almacenan de forma no volátil los valores de las variables o la configuración especifica del dispositivo, por ejemplo, el modo conexión WIFI o las alarmas RTC programadas, de tal forma que dichos valores de configuración puedan ser cargados de nuevo tras pasar por los estados de suspensión o hibernación. Esta memoria está dividida en dos grupos de direcciones.

  - Direcciones 0-1023.Reservado para los valores de fábrica. No es recomendable, escribir sobre este espacio.

  - Direcciones 1024-4095. Usado para guardar variables y valores de configuración. Por tanto, realmente se dispone de 3KB para el uso del programa cargado.

- **Memoria SRAM (8KB).** Usado para almacenar los valores de las variables durante la ejecución del programa. Es posible realizar una reserva y liberación dinámica de memoria, pero no es recomendable ya que no es posible utilizar una herramienta cómo Valgrind [38] para depurar las fugas de memoria. La razón por cual no puede ser utilizada se debe al hecho que la ejecución del programa se ejecuta cíclicamente desde el propio dispositivo y no desde un terminal del usuario o desde el entorno del desarrollo de Waspmote. Independientemente, de la herramienta utilizado, la documentación ofrecida por Libelium recomienda no utilizar reserva y liberación dinámica de memoria[referencia].

- **Memoria SD (max. 2GB).** En ella, se pueden almacenar de forma estática cualquier fichero o directorio empleando un sistema de ficheros FAT16. Por otro lado, el dispositivo no suporta tarjetas de más de 2GB.

5. 1. 2. Módulo WiFi

Aunque Waspmote suporta una amplia gama de interfaces inalámbricas (Bluetooth, 3G, ZigBee...) y configuraciones posibles, en el proyecto se ha optado por una interfaz Wifi. La siguiente tabla resume las características básicas [29].

Tabla 2. Hoja de características del módulo WiFi

Datos Generales	**Protocolo:**	802.11b/g - 2.4GHz
	Potencia TX:	0dBm - 12dBm
	Sensibilidad RX Sensitivity:	-83dBm
	Conector antena:	RPSMA
	Antena:	2dBi / 5dBi
	Seguridad:	WEP, WPA, WPA2
	Topologías:	AP / Adhoc
Ancho banda	**Transmisión:**	Baud: 9600/19200/57600 kbps: 5,63 / 9,19 / 15,68
	Recepción:	Baud: 9600/19200/57600 kbps: 100 / 100 / 100
Consumos	**OFF**	0 uA
	SLEEP	4 uA
	ON	33 mA
	Tx&Rx Data	38 mA
	Scanning Access Points	34 mA
Cobertura	**Antena 2dBi**	50-100m
	Antena 5dBi	300-500m
Tiempos de conexión	**AP**	4s
	Ad-hoc	10s

A modo de resumen, se ha recopilado la siguiente información acerca de las posibilidades de interfaces inalámbricas que se pueden usar con las Waspmotes [32]:

Tabla 3. Comparativa de módulos de comunicación Waspmote

	Protocolo	**Frecuencias**	**Potencia TX**	**Sensibilidad RX**	**Cobertura**
Xbee-805.15.4	805.15.4	2,4Ghz	0dBm	-92dB	500m
Xbee-805.15.4-Pro	805.15.4	2,4Ghz	20	-100dBm	7000m
Xbee-ZB	ZigBee-Pro	2,4Ghz	3dBm	-96dBm	500m
Xbee-ZB-Pro	ZigBee-Pro	2,4Ghz	17dBm	-102dBm	7000m
Xbee-868	RF	868Mhz	25dBm	-112dBm	12Km
Xbee-900	RF	900Mhz	17dBm	-100dBm	10Km
Wifi	802.11b/g	2,4/5,4Ghz	0-12dBm	-83dBm	50-100m(Antena 2dBi) 300-500m(Antena 5dBi)
Bluetooth	Bluetooth v2.1+EDR Class2		3dBm		
GSM/GPRS	SIM900	850/900/1800/1900Mhz	1-2W	-109dBm	

5. 2. Aspectos Software

Una vez se ha descrito las características y composición hardware de los nodos *Waspmote*, en este apartado se explicarán todos los aspectos de implementación en un entorno programación *Waspmote* [30] y las limitaciones que han motivado modificaciones en la implementación del diseño descrito en el apartado 4.

5. 2. 1. Estructura del código fuente

El programa a cargar en dispositivo Waspmote emplea un lenguaje de programación C/C++ y consta de las siguientes secciones.

- **Carga de librerías, definición de constantes y declaración de variables globales.** Empleando las instrucciones *#include* y *#define*, respectivamente.

- **Función void setup ().** Ejecutada una vez durante la carga del programa en el dispositivo, o bien, en durante el arranque del dispositivo. En esta función, se implementaran las instrucciones necesarias para la inicialización de los módulos Waspmote (WiFi, Sensores, RTC...). Estos valores serán almacenados en la EEPROM de tal forma que si el dispositivo entra en estado de suspensión o hibernación, pueda recuperarse la configuración inicial.

- **Función void loop ().** Se ejecuta indefinidamente en bucle a lo largo del ciclo de vida del programa. En esta función, implementa la instrucciones de lectura de sensores, envío y recepción de mensajes, manejo de ficheros en la memoria SD....

Figura. 15. Estructura del Código Fuente

```
// 1. Include Libraries
// 2. Definitions
// 3. Global variables declaration
void setup()
{
// 4. Modules initialization
}
void loop()
{
// 5. Measure
// 6. Send information
// 7. Sleep Waspmote
}
```

5. 2. 2. Ciclo de vida del programa

Cómo ya se describió en el apartado anterior**,** hay dos funciones principales en el programa cargado en dispositivo, una ejecutada una vez para inicializar la configuración y otra que será ejecutada en bucle y que realmente define el funcionamiento del dispositivo en la aplicación correspondiente. Sin embargo, el concepto de un programa ejecutándose en un bucle infinito plantea una cuestión importante en cuanto al consumo energético.

Para resolverlo, Waspmote define una serie de modos de operación e interrupciones para la placa madre y los módulos insertados en ella.

5. 2. 2. 1. Modos de operación

La placa de Waspmote cuenta con 4 modos de operación cada uno caracterizado por su consumo energético, el intervalo de tiempo que pueden permanecer en dicho estado y las interrupciones que permiten despertar al dispositivo.

Tabla 4. Modos de operación del dispositivo

Modo de operación	**Consumo**	**Estado del microcontrolador**	**Duración del modo de operación**	**Interrupciones que activan el dispositivo**
ON	15mA	Activo	-	Todos las interrupciones
SLEEP	55µA	Activo	De 32ms a 8s	Todas la interrupciones asíncronas Interrupciones síncronas del WDT
DEEP SLEEP	55µA	Activo	De pocos segundos a varios días	Todas la interrupciones asíncronas Interrupciones síncronas del RTC
HIBERNATE	0,06µA	Apagado	De pocos segundos a varios días	Interrupciones síncronas del RTC

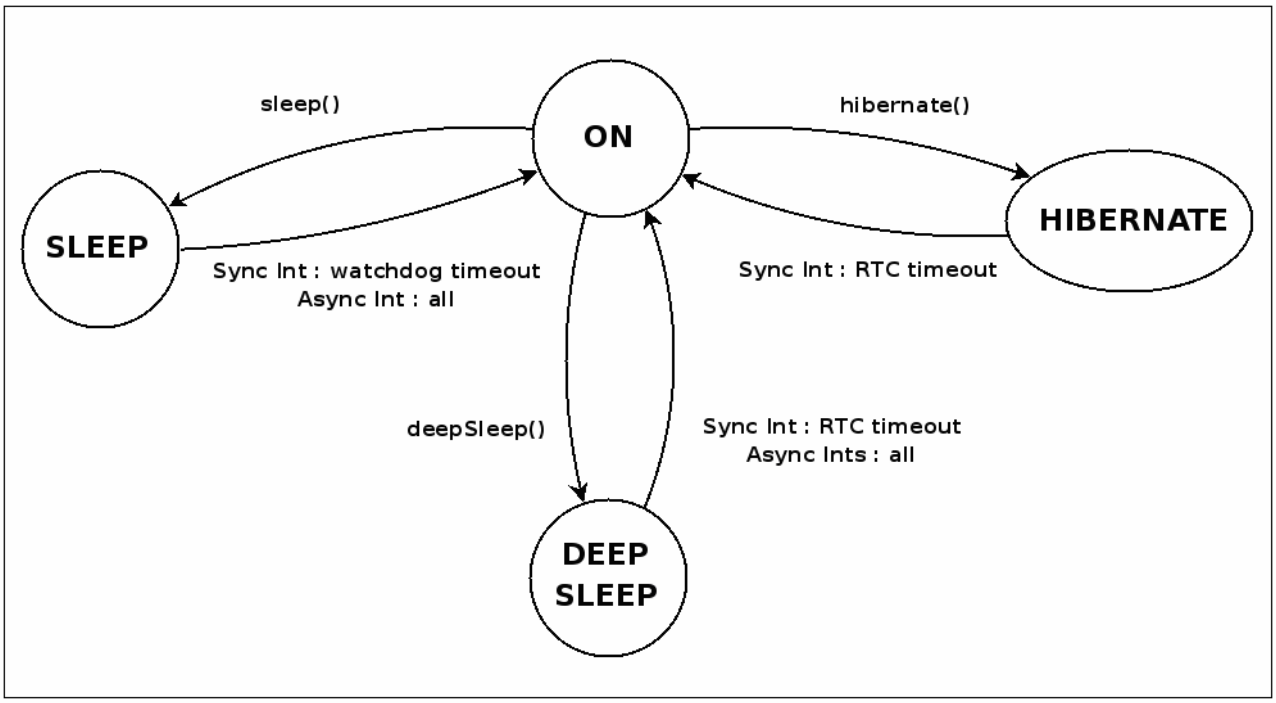
Por otro lados, los módulos insertados en la placa Waspmote tienen sus propios modos de operación, los cuales son dependientes de los modos de operación de la placa principal.

- **ON.** Modo de operación normal.
- **Sleep.** Algunas funciones del módulos son detenidas y pueden ser reactivadas de forma asíncrona.
- **Hibernate.** Todas funciones del módulos son detenidas y pueden ser reactivadas de forma asíncrona.
- **OFF.** Cualquier módulo puede ser apagado desde el programa principal.

5. 2. 2. 2. Interrupciones

Las interrupciones son eventos generados de forma síncrona o asíncrona que reactivan el dispositivo. El uso de un tipo u otro de interrupción depende del modo de operación en el que se encuentre el dispositivo y la naturaleza de la misma. En la figura 16, puede observarse la interacción entre distintas interrupciones y los modos de operación.

Figura. 16. . Modos de operación e interrupciones [30]



- **Interrupciones síncronas.** Son controladas por los temporizadores RTC (Real Time Clock) y WDT (WatchDog Timer) y pueden subdividirse en dos tipos.
  - Alarmas periódicas, controladas por el RTC y que son activadas al alcanzar la fecha y hora programadas.
  - Alarmas relativas, controladas por el RTC o el WDT y que son activadas cuando el contador programado expira.
- **Interrupciones asíncronas.** Son no programables y solo pueden ser generadas por ciertos módulos del dispositivo:
  - Módulos sensor. Lanzan una interrupción, por ejemplo, cuando la temperatura alcanza cierto valor o se detecta movimiento.
  - Acelerómetro. Es un sensor integrado la placa principal que detecta cambios en la velocidad o impactos.
  - módulo Xbee. A través del protocolo Digimesh, es posible dormir y activar el dispositivo de forma cíclica.

5. 2. 3 Temporizadores

Wapmote emplea dos mecanismos para generar interrupciones síncronas:

- **WDT (Watchdog Timer).** En este mecanismo se establece un intervalo de tiempo y cuando el contador alcanza dicho intervalo, lanza una interrupción la cual puede ser procesada en los modos de operación ON y SLEEP. Los intervalos definidos son 16ms, 32ms, 64ms, 128ms, 256ms, 500ms, 1s, 2s, 4s y 8s. A partir de 8s, el dispositivo pasa al modo DEEP SLEEP y han de utilizarse la interrupciones del RTC.

- **RTC (Real Time Clock).** Este mecanismo genera interrupciones en base a una fecha y absolutos, lo cual implica configurar la fecha y hora del dispositivo y de las alarmas que generan las interrupciones. El reloj RTC y sus alarmas permanecen activos en todos los modos de operación por lo que puede ser utilizado cuando el dispositivo esta activo (On) o para despertarlo en uno de los modos de operación de bajo consumo (Sleep, DeepSleep o Hibernate).

6. Prototipo de sistema CCN para Waspmotes

El objetivo de esta implementación no es crear sistema completamente funcional basado CCN, sino desarrollar la aplicación de nuestro diseño de CCN sobre una plataforma real de red de sensores como Waspmote.

Una vez explicado el paradigma CCN, las características de las redes WSN, el diseño final para la implementación del módulo CCN-WSN de este proyecto y los aspectos técnicos más relevantes del dispositivo Waspmote. En este capítulo, describiremos la implementación de un prototipo CCN sobre Waspmotes y el código desarrollado. También se comentarán las limitaciones de los dispositivos Waspmotes que han condicionado el prototipo.

6. 1. Limitaciones de dispositivo

Al contrario que otros equipos de comunicaciones más complejos, los nodos Waspmote presentan mayores restricciones que no pueden ser solventadas o cuya solución está fuera de los objetivos de este proyecto.

A) Gestión del tiempo en los dispositivos

Tal y cómo se explicó en el apartado *5. 2. 3 Temporizadores*, existen dos tipos de temporizadores, WDT y RTC, los cuales han sido necesarios para definir las marcas de tiempo de los mensajes *ContentObject* y las tablas *ContentStore* y *PIT*. Particularizando cada caso, los obstáculos surgidos en este aspecto han sido:

- En el caso del temporizador WDT, los intervalos de tiempo están restringidos a valores discretos no mayores que 8 segundos (16ms, 32ms, 64ms, 128ms, 256ms, 500ms, 1s, 2s, 4s y 8s). Esto limita los valores permitidos en los campos *InterestLifeTime* y *FreshnessSeconds*, así cómo, los temporizadores de las tablas *ContentStore* y PIT.

- En el caso del reloj RTC, es necesario inicializar la fecha y hora de los dispositivos manualmente antes de poder ser utilizados. Esto presenta dos problemas importantes:

    - La precisión del reloj puede ser cómo máximo de segundos.

- Se podría, en general, realizar una sincronización perfecta de la fecha y hora en múltiples dispositivos mediante el servicio NTP (*Network Time Protocol*). Pero la implementación de esta solución ha quedado fuera de los objetivos de este proyecto.

Para resolver los problemas de WDT se optó por excluir las interrupciones WDT de este proyecto y emplear las alarmas del RTC en su lugar. Por otro lado, los problemas de sincronización entre los dispositivos se mitigan estableciendo un configuración inicial del temporizador RTC (3/jun/2014 00:00:00) y arrancando los dispositivos al mismo tiempo. Sin embargo esta solución no resuelve el hecho de que si un nodo es reiniciado, pierde todo sincronismo con la red CCN-WSN. Una solución definitiva pasaría por el uso de NTP, al menos en el momento de arranque del dispositivo.

B) Interrupciones y modos de operación

El propósito de las interrupciones y los modos de operación, tal y cómo se indicó en los apartados 5. *2. 2. 1. Modos de operación* y *5. 2. 2. 2. Interrupciones*, es reducir el consumo energético del dispositivo. Sin embargo, los modos de operación Sleep, DeepSleep o Hibernate no solo exigen conservar el valor de las variables en la EEPROM o en la tarjeta SD de forma explícita, sino que también desactivan el módulo WiFi provocando la perdida de mensajes.

Debido a ello, en nuestro caso, el dispositivo ha de permanecer activo durante toda la ejecución y únicamente podrá utilizar las interrupciones síncronas del RTC.

C) Concurrencia

El microcontrolador ATmega 1281 es incapaz de ejecutar múltiples procesos o hilos de ejecución. Este hecho ha condicionado enormemente la estructura y funcionamiento del programa y ha imposibilitado la implementación de otros procesos como, por ejemplo:

- Procesos separados para las funciones de recepción y transmisión.
- Procesos que interactúen con el usuario que ejecuten la petición Interest por parte de un Face de aplicación y la entrega final de los contenidos.
- Procesos que permitan monitorizar los paquetes cómo tcpdump.
- Procesos que afecten a la transmisión cómo protocolos de enrutamiento o peticiones del protocolo NTP.

D) Gestión de memoria RAM

En el apartado *5. 1. 1. 1. Elementos de Almacenamiento*, describimos las diferentes memorias del dispositivo Waspmote y cómo limitaban las dimensiones del código fuente y del conjunto de variables que podía usar el programador. Para dar perspectiva, la implementación final de este proyecto ha supuesto un consumo del 26% del la memoria Flash (tamaño del código) y un consumo del 87% de la memoria RAM (variables). Al reservar todo el espacio de las variables de forma estática, el sistema es ineficiente, por ejemplo, un *ContentStore* representado por un array fijo de N entradas puede estar infrautilizado, pero tampoco puede ser extendido durante la ejecución del programa.

La reserva y liberación de memoria dinámica resolvería este problema de eficiencia, sin embargo, no hemos implementado estos mecanismos ya que el dispositivo Waspmote no puede ejecutar herramientas cómo valgrind [38] en tiempo de ejecución para detectar y depurar fugas de memoria aunque las funciones cómo malloc, calloc o free sí estén disponibles.

E) Declaración de variables y manejo de punteros

En el lenguaje de programación C/C++, al declarar una variable se reserva un espacio en la RAM y dicho espacio es accesible a través de un puntero. Estas variables, ya declaradas, pueden ser utilizadas como parámetros o para contener los resultados de las funciones. En el caso de las variables complejas, es necesario pasar el puntero cómo un parámetro de la función y copiar el resultado explícitamente mediante las funciones memcpy o strcpy [14][33].

La limitación del dispositivo a la hora manejar estos punteros consiste en que, aun utilizando las funciones memcpy o strcpy, los valores no se conservan en los punteros pasados cómo parámetros. Para resolverlo, fue necesario declarar dichos punteros cómo variables globales de tal forma que sus espacios de memoria sean accesibles en todo el programa (tal y cómo recomienda la guía de programación de Waspmote)[30].

F) Versión de los dispositivos Waspmote

A día de hoy, existen dos versiones del dispositivo Waspmote, Waspmote V1.1 y Waspmote Pro V1.2. Aunque sus especificaciones técnicas son muy similares ciertas funciones y librerías difieren entre si [30] lo cual implicaría desarrollar dos versiones del código. Debido a este hecho, solamente se ha desarrollado la versión para Waspmote Pro V1.2.

G) Cálculo de SHA256 para el campo Signature

A lo largo de la implementación proyecto, ha sido necesario calcular el valor SHA256 para la componente SHA256 del *ContentName* de los mensajes *Interest*, los campos *PublisherPublicKeyDigest* de los mensajes *Interest* y *ContentObject* y el campo *SignatureBits* del mensaje *ContentObject*. Analizando brevemente la función sha de la librería WaspHash.h (*void sha(uint8_t alg,uint8_t hash_message[],uint8_t* message,uint32_t size_message)*), esta guarda el resultado en el parámetro *hash_message* leyendo byte a byte del valor contenido en el parámetro *message* hasta encontrar el byte 0x00.

Este hecho presenta un problema importante a la hora de calcular la firma digital del *ContentObject*, ya que al aplicar la función *sha* determinados campos de *SignedInfo* como *PublisherPublicKeyDigest* o *Type*, estos pueden contener el valor 0x00 y daría un valor SHA256 incorrecto.

Para resolver este problema, se han obviado los campos conflictivos y, por tanto, el cálculo de la firma queda limitado a los siguientes datos del *ContentObject*:

- *ContentName*. Excluyendo la marca '\0' al final del valor.
- *SignedInfo*: *Timestamp* (excluyendo la marca '\0' al final del valor) y *FreshnessSeconds*.
- Contenido. Siendo este último el que contiene la marca '\0' al final de la secuencia.

Alternativamente, se podría haber programado una nueva versión de la función *sha* para no depender de la marca '\0', pero lo hemos dejado fuera de los objetivos del proyecto.

H) Retardo de las funciones de transmisión y recepción de mensajes

Durante el desarrollo de las funciones de transmisión y recepción de mensajes se observaron dos hechos que influencian la aplicación de las mismas.

- El retardo en la apertura del socket puede llegar a varios segundos, lo cual implica una posible pérdida de paquetes en la recepción y análisis de los mensajes.
- No es posible la transmisión de dos mensajes consecutivos. Ello implica forzar un retardo entre transmisiones, o bien, reabrir el socket en cada transmisión.

6. 2. Implementación software del módulo CCN-WSN

En este apartado, describiremos las estructuras de datos y variables globales, la inicializaron de las mismas, el cuerpo del programa y las funciones auxiliares empleadas a lo largo del programa.

6. 2. 1. Constantes principales, estructuras de datos y variables globales

Dada la limitación de espacio en la memoria RAM, fue necesario modificar el tamaño de las estructuras de datos y de las variables globales a lo largo del desarrollo del proyecto, por lo que el uso de constantes fue clave para facilitar estos cambios, por ejemplo, emplear las constantes MAX_CN_COMP_SIZE y MAX_CN_COMP_NUM todas las funciones que manejen los ContentName permite redimensinar el tamaño de estas variables sin tener que modificar los mecanismos relacionadas con ellas.

6. 2. 2. Inicialización del programa

El bloque de funciones de inicialización se ejecuta dentro de la función *setup*, que tal y cómo se describió en el apartado *5. 2. 1. Estructura del código fuente*, solamente se ejecuta una vez tras la carga del programa o tras el arranque del dispositivo. Las funciones de inicialización pueden ser divididas en dos categorías:

- Inicialización los módulos del dispositivo. En esencia el módulo Wifi, el temporizador RTC y el puerto USB para la comunicación por consola.

    - *init_wifi()*. Consiste básicamente en la activación del módulo WiFi, configuración IP (IP, mascara de red e IP de puerta de enlace), configuración del protocolo de transporte (protocolo UDP y puerto de origen) y asociación con el punto de acceso.

    - *init_RTC()*. Activa el módulo RTC e inicializa el reloj al valor contenido en la constante SETUP_RTC a través de la función *setTime()* de la librería WaspRTC.

    - *init_usb()*. Únicamente contiene una llamada a la función *USB.ON()*. Una vez activada la interfaz USB se imprime por pantalla mediante llamadas a las funciones USB.print (), USB.println () y USB.printHex (). De forma complementaria, la función USB.OFF() desactiva la interfaz USB impidiendo la posibilidad de imprimir mensajes por pantalla. En un despliegue real y permanente de Waspmotes recomendable eliminar todas las llamadas a funciones del impresión del código fuente reduciendo de esta forma el tamaño del código fuente y el consumo de memoria RAM.

- Inicialización de las variables de CCN.

  - Lista de *Faces*, Inicializada a través de las funciones void init_face (uint8_t face_id, char * host, uint8_t ip_proto, int port) y void init_face_list ().

  - *ContentStore*. Inicializada a través de las funciones *void init_contentstore(), void delete_entry_cs (int idx), void init_file_system()* y *void init_content ()*. Esta última carga una lista de ficheros ficticios, es decir, una lista de nombres de contenido en el *ContentStore*.

  - Tabla PIT. A través de la función *void init_pit()* y *void delete_entry_pit (int idx)*. Inicialmente la tabla PIT estará vacía.

  - Tabla FIB. A través de la función *void init_fib()*. Dado las que no se ha implementado ningún protocolo de gestión de Faces o diversidad reglas de estrategia, la tabla FIB habrá de contener las entradas fijas, tal y como nuestra la siguiente tabla. Cabe decir que los valores del campo *StrategyRuleID* resultan triviales dado que solamente se ha definido una regla de estrategia.

Tabla 5. Tabla FIB programada

Prefix	**StrategyRuleID**	**Description**	**OutputFaces**
"/"	1	Broadcast	1
"/ccnd"	2	Remote Broadcast	2
"/ccnd/B"	3	Unicast	3
"/ccnd/C"	4	Unicast	4

6. 2. 3. Cuerpo del programa

Una vez se han definido los elementos del programa y los estados iniciales de los mismos, procedemos a explicar el comportamiento del programa definido dentro de la función *loop* que, tal y como explicamos en el apartado *5. 2. 1. Estructura del código fuente*, se ejecuta cíclicamente. Para ello, se programaron tres grandes funciones:

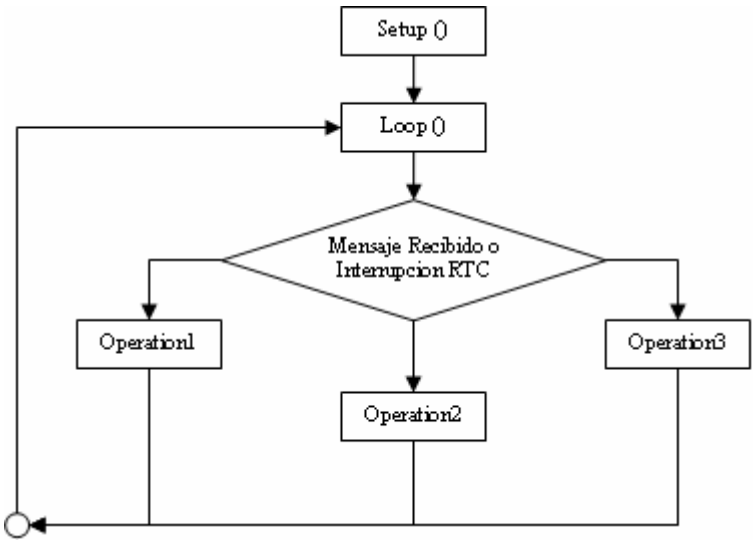
- *void operation1 ().* Función que define el comportamiento de programa ante la llegada mensaje *Interest*.

- *void operation2 ().* Función de define el comportamiento de programa ante la llegada mensaje *ContentObject*.

- *void operation3 ().* Función encargada del reenvío periódico de mensajes *Interest* pendientes contenidos en la tabla PIT.

El siguiente diagrama flujo muestra la lógica general del programa y, en posteriores subapartados, explicaremos individualmente operation1, operation2 y operation3.

Figura. 17. Diagrama flujo general del programa



Otro punto clave a tener en cuenta es el mecanismo que define qué operación es ejecutada. Dicho mecanismo ha de basarse en la recepción de mensajes (*operation1()* y *operation2()*) y en el intervalo de tiempo para la consulta periódica de la tabla PIT (*operation3()*). Desgraciadamente, no pudo completarse la función encargada de la recepción de mensajes, por lo que en las pruebas descritas en el apartado *6. 3. Pruebas del Sistema*, hubo que generar los mensajes *Interest* y *ContentObject* de forma ficticia.

No obstante, la posible implementación de este mecanismo pasaría por utilizar el mismo intervalo tiempo para la escucha del socket de recepción y para la comprobación de la tabla PIT. Basándonos en un código de ejemplo proporcionado por Libelium, *WIFI_22_UDP_client* [34][35], el siguiente fragmento de código sería la solución.

```
do
{
        //AbrirSocket
        WIFI.setUDPclient(IP_ADDRESS, REMOTE_PORT, LOCAL_PORT);
        if(WIFI.read(NOBLO)>0) // Lectura del socket
        {
                if (WIFI.length == sizeof (InterestMsg))
                {
                        operation1();
                }
                else if (WIFI.length == sizeof (ContentObjectMsg))
                {
                        operation2();
                }
        {
        WIFI.close(); // Cerrar Socket
} while (experied_inteval());
operation3 ();
```

6. 2. 3. 1. Función Operation1

La ejecución de esta operación implica la siguiente secuencia de pasos, consecuentes con el procedimiento del mensaje *Interest* descrito en el apartado *4. 3. 4. Procesamientos de mensajes CCN-WSN.*

0. Obtención del mensaje *Interest*. Dado que en la implementación final del proyecto no es posible recibir mensajes *Interest*, los propios mensajes *Interest* son generados usando la función *generate_interest* que describiremos más adelante.

1. Verificación del mensaje *Interest*. Se comprueba el formato y el valor de la componente SHA256 del *ContentName* y el campo *PublisherPublicKeyDigest*. El único formato válido para el *ContentName* de un mensaje *Interest* es *"/Comp1/Comp2/.../CompN/CompSHA256"*. Por supuesto, si alguna de estas comprobaciones falla, el mensaje *Interest* es descartado y la ejecución de operation1 finaliza. Las funciones utilizadas en este paso son: *split_content_name*, *generate_SHA256*, *check_contentname*, *size_contentname* y *check_SHA256.*

2. Búsqueda del *ContentName* en el *ContentStore*. A través de la función *preferent_order*, primero se excluyen aquellas entradas que no contienen el prefijo demandado o que están vacías; segundo, se ordenan las posiciones del resto de las entradas siguiendo las directrices del orden canónico; y, finalmente, dependiendo del valor del campo *ChildSelector*, las posiciones de las entradas son leídas en un sentido o el opuesto para seleccionar el *ContentObject* que mejor satisfaga el mensaje *Interest*. El resultado de esta búsqueda puede ser:

- Encontrar un *ContentObject* que satisfaga el *Interest.* Por lo que, las siguientes tareas a realizar son: generar el *ContentObject*, enviarlo por la *Face* de entrada del mensaje *Interest* y actualizar el campo *LastTimeUsed* de la entrada en el *ContentStore*. Una vez realizadas estas tareas, la función operation1 finaliza.

- No encontrar un *ContentObject* que satisfaga el *Interest* y continuar el proceso de *operation1* en la tabla PIT.

Las funciones aplicadas en este paso han sido: *manage_cs*, *search_content_store*, *preferent_order* y *send_content_object*.

3. Búsqueda del ContentName en el PIT. En este paso, se buscan aquellas entradas en la PIT cuyo prefijo coincida exactamente con el *ContentName* de mensaje *Interest* y cuyo valor en el campo *RequestedFace* coincida con la *Face* de entrada del mensaje *Interest*. El resultado de esta búsqueda puede ser:

- Encontrar una entrada en la PIT con este criterio, en cuyo caso el proceso de operation1 finaliza.

- No encontrar ninguna entrada en la PIT con este criterio. Por lo que, las siguientes tareas a realizar son: añadir una nueva entrada en la PIT y continuar el proceso de operation1 en la tabla FIB.

Las funciones aplicadas en este paso han sido: *manage_pit*, *search_pit* y *preferent_order*.

4. Búsqueda del *ContentName* en la FIB. En esta búsqueda, se aplica el criterio *Longest Prefix Match* ordenando las entradas de la FIB para establecer un orden de preferencia. El orden de preferencia establecido es:

- Primero aquella entrada que tenga coincidencia perfecta en longitud y en contenido con el prefijo demandado en el mensaje *Interest.*

- Segundo aquellas entradas que contengan parte del prefijo demandado en el mensaje *Interest*.

- Tercero, siguiendo un orden decreciente componente a componente hasta alcanzar la compone nula "/".

Por ejemplo, si el *Interest* demanda el *ContentName "/ccnd/C"* y la tabla contiene los prefijos *"/"*, *"/ccnd"*, *"/ccnd/B" "/ccnd/C"* y *"/ccnd/C/Fol1"*, el orden de preferencia es *"/ccnd/C"*, *"/ccnd/C/Fol1"*, *"/ccnd"* y *"/"*, excluyendo la entrada *"/ccnd/B"*. Siguiendo este orden de preferencia, el mensaje *Interest* es enviado por las Faces de salida indicadas por la entrada de la FIB seleccionada y los campos *Tries* y *LastTimeTried* de la entrada de la tabla PIT son actualizados. Llegados a este punto, el proceso de operation1 finaliza.

Las funciones aplicadas en este paso han sido: *search_fib, preferent_order* y *send_interest.*

6. 2. 3. 2. Función Operation2

La ejecución de esta operación implica la siguiente secuencia de pasos, consecuentes con el procedimiento del mensaje *ContentObject* descrito en el apartado *4. 3. 4. Procesamientos de mensajes CCN-WSN.*

0. Obtención del mensaje *ContentObject*. Al igual que en caso de operation1, los mensajes *ContentObject* no son recibidos sino que son generados usando la función *generate_content_object* que describiremos más adelante.

1. Verificación del mensaje *ContentObject*. Se comprueba el valor de la firma digital y el valor del campo *PublisherPublicKeyDigest*. Por supuesto, si alguna de estas comprobaciones falla, el mensaje es descartado y la ejecución de operation2 finaliza. Las funciones aplicadas en este paso han sido: *check_signature*, generate_SHA256 y *check_SHA256.*

2. Búsqueda del *ContentName* en el *ContentStore*. En este paso, se buscan aquellas entradas en el *ContentStore* cuyo *ContentName* coincida exactamente el *ContentName* de mensaje *ContentObject*. El resultado de esta búsqueda puede ser:

- Encontrar una entrada en el *ContentStore* con este criterio, en cuyo caso se actualizan los campos de la entrada en el *ContentStore*, es decir, el valor de *StalenessBit* es desactivado, los valores de *Timestamp* y *FreshnessSeconds* del *ContentObject* son copiados respectivamente en los campos *CreationTimestamp* y *FreshnessSeconds* del *ContentStore*, y, finalmente el campo *LastTimeUsed* del *ContentStore* es borrado. Realizadas estas tareas, el proceso de operation2 finaliza.

- No encontrar ninguna entrada en el *ContentStore* con este criterio. Por lo que, las siguientes tareas a realizar son: añadir una nueva entrada en el *ContentStore* y continuar el proceso de operation2 en la tabla PIT.

Las funciones aplicadas en este paso han sido: *manage_cs*, *search_content_store* y *preferent_order.*

3. Búsqueda del *ContentName* en la tabla *PIT*. En este paso, se buscan aquellas entradas en la PIT cuyo prefijo coincida con una parte del *ContentName* de mensaje *ContentObject* recibido.

- Encontrar una o varias entradas en la tabla PIT con este criterio, en cuyo caso se actualizan el campo *LastTimeUsed* de la entrada en el *ContentStore*, enviar una copia del mensaje *ContentObject* a través de la *Face* indicada en el campo *RequestedFace* de la entrada PIT y eliminar la entrada en la tabla PIT.

- No encontrar ninguna entrada en la tabla PIT con este criterio. En este caso, el proceso de operation2 finaliza.

Las funciones aplicadas en este paso han sido: *manage_pit*, *search_pit*, *preferent_order y send_content_object.*

6. 2. 3. 3. Función Operation3

El objetivo de operation3 es el reenvío periódico de los mensajes de *Interest* pendientes contenidos en la tabla PIT. Al igual que en apartados anteriores, describiremos esta operación como una secuencia ordenada de pasos.

1. Eliminar entradas antiguas en la tabla PIT. A través de la función *manage_pit*, se analiza la tabla PIT y se eliminan aquellas entradas que hayan excedido el valor del campo *InterestLifeTime* desde la fecha y hora indicadas en el campo *CreationTimestamp*.

La funciones aplicadas en este paso han sido: *manage_pit y expired_pit_entry.*

2. Búsqueda de *RequestedPrefix* en la tabla FIB. De forma similar al cuarto paso de operation1 descrito en el apartado *6. 2. 1. Función Operation1*, se establece el orden de preferencia de las entradas de la tabla FIB en base al valor contenido en el campo *RequestedPrefix* de entradas de la tabla PIT no eliminadas en el paso anterior, se generan los mensajes *Interest*, se envían siguiendo el orden de preferencia de la FIB y, finalmente, los campos *Tries* y *LastTimeTried* de la entrada de la tabla PIT son actualizados. Una vez realizadas estas tareas, la función operation3 finaliza.

Las funciones aplicadas en este paso han sido: *search_pit, search_fib y send_interest.*

6. 2. 3. 4. Funciones Auxiliares

A) Generación de mensajes

Los mensajes, definidos en las variables globales *interest* y *content_object*, pueden ser generados a través de las funciones *generate_interest* y *generate_content_object*, o bien, copiados desde el buffer del módulo WiFi. Esta última opción no pudo ser implementada finalmente, aunque las siguientes dos instrucciones hubieran sido claves para copiar los datos contenidos en el buffer de recepción.

*memcpy ((void *)&interest, (void *) WIFI.answer, sizeof (InterestMsg));*

*memcpy ((void *)&content_object, (void *) WIFI.answer, sizeof (ContentObjectMsg));*

No obstante, las funciones que sí fueron implementadas en este aspecto son:

- *void generate_interest (char * contentname, char WIFI.answer* public_key, uint8_t child_selector, uint8_t interest_life).* Establece los valores de los distintos campos de la variable global *interest* del tipo estructura *InterestMsg*.
- *void generate_content_object (char * contentname, char * public_key, char * timestamp, int freshness_seconds).* Establece los valores de los distintos campos de la variable global *content_object* del tipo estructura *ContentObjectMsg*.

B) Búsqueda de información en las tablas

En nuestra implementación, el mecanismo de búsqueda en las tablas principales de un nodo CCN-WSN, a saber, *ContentStore*, tabla PIT y tabla FIB, se fundamenta en la función *preferent_order* y en tres arrays de enteros que almacenan las posiciones absolutas de las entradas de las tablas definidas en las variables globales *content_store*, *pit_table* y *fib_table*.

- *int search_content_store (uint8_t mode).* Esta función realiza búsquedas en el *ContentStore* empleando la función auxiliar *preferent_order* y devolviendo un valor entero que indica si la búsqueda ha tenido éxito o no. El funcionamiento interno de esta función depende del valor del parámetro *mode* que selecciona uno de los siguientes modos de operación.
    - 0: Búsqueda en el *ContentStore* ante la llegada de un mensaje *Interest* (función principal *operation1*).
    - 1: Búsqueda en el *ContentStore* ante la llegada de un mensaje *ContentObject* (función principal *operation2*).

- 2: Búsqueda en el *ContentStore* para el mantenimiento de la propia tabla.

- *int search_pit (uint8_t mode).* Esta función realiza búsquedas en la tabla PIT empleando la función auxiliar *preferent_order* y devolviendo un valor entero que indica si la búsqueda ha tenido éxito o no. El funcionamiento interno de esta función depende del valor del parámetro *mode* que selecciona uno de los siguientes modos de operación.

  - 0: Búsqueda en la tabla PIT ante la llegada de un mensaje *Interest* (función principal *operation1*).

  - 1: Búsqueda en la tabla PIT ante la llegada de un mensaje *ContentObject* (función principal *operation2*).

  - 2: Búsqueda en la tabla PIT para el mantenimiento de la propia tabla.

- *int search_fib ().* Esta función realiza búsquedas en la tabla PIT empleando la función

auxiliar *preferent_order* y devolviendo un valor entero que indica si la búsqueda ha tenido éxito o no. Dado que la tabla FIB estática y que su utilización es muy limitada, no fue necesario definir diferentes modos de operación como en las funciones *search_content_store* y *search_pit*.

- *void preferent_order (char * contentname, uint8_t table).* Esta función aplica un criterio de selección determinado sobre una cadena de texto indicada en el parámetro *contentname* y una de las tres tablas indicada en el parámetro *table*. El resultado de esta función es un array de posiciones que indica que entradas del *ContentStore*, PIT o FIB cumplen un determinado criterio: *Longest Prefix Match*, orden canónico, coincidencia parcial o coincidencia completa.

C) Mantenimiento de la información en las tablas

El siguiente grupo de funciones son necesarias para mantener actualizada la información de la distintas tablas y por tanto, son empleadas en las tres funciones principales.

- *void manage_pit ().* Comprueba todas las entradas de la tabla PIT eliminando la entrada en función del valor devuelto por la función *expired_pit_entry*.

- *void manage_cs ().* Comprueba todas las entradas del *ContentStore* activando el campo *StalenessBit* o eliminando la entrada en función de los valores devueltos por las funciones *staleness_cs_entry* y clear_cs_entry.

- *int expired_pit_entry (int idx).* Comprueba si el tiempo de vida de una entrada en la tabla PIT ha expirado y retorna el valor uno, en caso afirmativo, y el valor cero, en caso contrario.

- *int staleness_cs_entry (int idx).* Comprueba si una entrada en el *ContentStore* ha de ser marcada como anticuada y retorna el valor uno, en caso afirmativo, y el valor cero, en caso contrario.

- *int clear_cs_entry (int idx).* Comprueba si una entrada en el *ContentStore* marcada como anticuada y que ha sido utilizada recientemente ha de ser eliminada y retorna el valor uno, en caso afirmativo, y el valor cero, en caso contrario.

- *void delete_entry_cs (int idx).* Borra una entrada concreta en el *ContentStore*.

- *void delete_entry_pit (int idx).* Borra una entrada concreta en la tabla PIT.

D) Funciones de transmisión y recepción

Las funciones relativas a la recepción de mensajes no pudieron ser completadas. No obstante, las funciones de transmisión de mensajes son completamente operativas.

- *void send_interest (uint8_t face_id).* Transmite el contenido de la variable global *InterestMsg* seleccionando la dirección IP y el puerto de destino asociadas a la *Face* indicada en el parámetro *face_id*. Internamente, las tareas realizadas en esta función son: apertura del socket, serialización el mensaje, envío del mensaje y cierre del socket.

- *void send_content_object (uint8_t face_id).* Transmite el contenido de la variable global *ContentObjectMsg* seleccionando la dirección IP y el puerto de destino asociadas a la *Face* indicada en el parámetro *face_id*. Internamente, las tareas realizadas en esta función son: apertura del socket, serialización el mensaje, envío del mensaje y cierre del socket.

E) Comprobación del *ContentName*

El siguiente grupo de funciones tiene como objetivo determinar la validez de un *ContentName* en lo que a su formato se refiere.

- *uint8_t size_contentname (char* contentname).* Cuenta el número de apariciones del carácter "/" en el ContentName. El hecho que de una secuencia SHA256 pueda contener el byte 0x2F (equivalente al carácter "/"), obliga a eliminar la componente SHA256 del *ContentName* a analizar.

- *int check_contentname (char* contentname).* Determina la validez del formato del ContentName, es decir, si es una secuencia de componente inferior a la constante MAX_CN_COMP_NUM y en la que cada componente es una secuencia de caracteres inferior a la constante MAX_CN_COMP_SIZE. Para determinar la validez de formato, se emplea el valor devuelto por la función:
  - 0: *ContentName* valido.
  - 1: *ContentName* invalido, debido a que el *ContentName* excede el número máximo de componentes.
  - 2: *ContentName* invalido, debido a que una alguna componente excede el número máximo de caracteres.
  - 3: *ContentName* invalido, debido a que una alguna componente está vacía, es decir, la secuencia "//" aparece en el *ContentName*.
  - 4: *ContentName* invalido, debido a que el *ContentName* finaliza con el carácter "/".
  - 5: *ContentName* invalido, debido a que el *ContentName* no comienza con el carácter "/".

F) Generación y comprobación SHA256

El cálculo y verificación de los valores SHA256 depende la exclusivamente de la función sha de la librería WaspHash.

void sha(uint8_t alg,uint8_t hash_message[],uint8_t message,uint32_t size_message);*

En todos los casos, el valor de los parámetros *alg* y size_message han sido las constantes SHA256 y SIZE_SHA256 (16 bytes), respectivamente.

Por otra lado, han sido necesarias las siguientes funciones para generar o comprobar los valores SHA256.

- *void generate_SHA256 (char * base). Función* genérica para crear secuencias de bytes SHA256.
- *uint8_t check_SHA256 (char * base).* Genera una secuencia SHA256 y comprueba el resultado con el valor SHA256 almacenado en una variable global. El resultado devuelto es cero, si la secuencia es correcta, y uno, en caso contrario.

- *void add_SHA256_component (char * contentname).* Genera una componente SHA256 en base al *ContentName* y añade el resultado al final del propio *ContentName*.

- *void split_content_name(char * contentname).* Separa la componente SHA256 de un ContentName. Por ejemplo, el *ContentName* "*/ccnd/A/56AB97FF2F56AB9700FF2F56AB97FF2F*" será dividido en "*/ccnd/A*" y "*56AB97FF2F56AB9700FF2F56AB97FF2F*".

- *void generate_signature ().* Genera la firma digital de los *ContentObject* a partir del *ContentName*, de los campos *Timestamp* y *FreshnessSeconds* de *SignedInfo*, y del propio contenido.

- *int check_signature ()*. Genera una firma digital a partir de un *ContentObject* y comprueba el resultado con el valor SHA256 almacenado en el campo *SignatureBits* del propio *ContentObject*. El resultado devuelto es cero, si la firma es correcta, y uno, en caso contrario.

G) Formato de las marcas de tiempo RTC y alarmas

La librería *WaspRTC* no tiene ninguna función que devuelva las marcas de tiempo en un formato numérico fácilmente manejable. En su lugar, las funciones de esta librería manejan cadena de texto en dos formatos.

- **Formato codificado**. "*YY:MM:DD:dow:hh:mm:ss*", usado cómo parámetro a la hora de establecer el fecha y hora del reloj RTC.

- **Formato legible**."*DoW, YY/MM/DD, hh:mm:ss*", devuelto por las funciones *getTimestamp* y *getTime* de la librería RTC.

El hecho que todas las marcas de tiempo tenga formato de cadena de texto ha requerido programar funciones que manejen estos formatos.

- *int diff_seconds (char * typed_timstamp1, char * typed_timstamp2).* Devuelve la diferencia en segundos de dos marcas de tiempo en formato codificado.

- *void convert_timestamp (char * time)*. Convierte una marca de tiempo en formato legible a una marca de tiempo en formato codificado.

6. 3. Pruebas del Sistema

El plan de pruebas del programa que expondremos a continuación muestra la ejecución del programa ante distintas situaciones. Al igual que en el apartado *6. 2. 3. Cuerpo del programa*, es posible dividir las pruebas en tres grandes grupos: tests de operation1, tests de operation2 y tests de operation3.

Finalmente analizando el resultados de las pruebas podemos concluir que la funcionalidad en el plano de CCN-WSN propuesta en el apartado 4 es completa.

6. 3. 1. Test de Operation1

A) Test 1 de operation1

- **Objetivo de la prueba.** Comprobar la organización del ContentStore siguiendo el orden canónico y verificar el mecanismo de selección de contenidos *ChildSelector*.
- **Estado inicial:** El *ContentStore* contiene múltiples entradas, todas ellas con el bit de antigüedad desactivado.

```
* CONTENT STORE
* Entry 0: /ccnd/A/file1
* Entry 1: /ccnd/A/Fol1/Fol2/file2
* Entry 2: Empty
* Entry 3: /ccnd/B/fil4
* Entry 4: /ccnd/A/Fol2/file1
* Entry 5: /ccnd/A/Fol2/file3
* Entry 6: /ccnd/A/Fol2/file2
* Entry 7: /ccnd/A/Fol2/file4
* Entry 8: /ccnd/A/Fol2/file0
* Entry 9: /ccnd/A/file2
```

- **Ejecución:** Recepción del primer mensaje *Interest* y entrega del *ContentObject "/ccnd/A/Fol2/file0*". Recepción del segundo mensaje *Interest* y entrega del *ContentObject "/ccnd/A/Fol2/file4*".

LOG: Tue, 14/06/03, 00:00:00 * Received Interest (Opt 1) by Face 1 * Interest Message ** ContentName: "/ccnd/A/Fol2/D6C62F3FA327916424ADCA4A7A6C13B6" ** PPKD: D22B6FDCC90777469FF9815F7C0E0026 ** ChildSelector: Leftmost ** InterestLifeTime: 10 ** /ccnd/A/Fol2/D6C62F3FA327916424ADCA4A7A6C13B6 Correct format and SHA256 component. ** D22B6FDCC90777469FF9815F7C0E0026 Correct PPKD. * Build ContentObject from entry 8 * Entry 8 of CS: ** ContentName: /ccnd/A/Fol2/file0 ** Type: DATA ** StalenessBit: OFF ** CreationTimestamp: Tue, 14/06/03, 00:00:00 ** FreshnessSeconds: 60 ** Last Time Used: Tue, 14/06/03, 00:00:01 * ContentObject Message ** Signarute: ** DigestAlgorithm: SHA256 ** Witness:IndSignature ** Signature Bits: 56EF2111188438D4DF0FD7E6C08E1F93 ** ContentName: "/ccnd/A/Fol2/file0" ** SignedInfo: ** PPKD: D22B6FDCC90777469FF9815F7C0E0026 ** Timestamp: Tue, 14/06/03, 00:00:00 ** Type: DATA ** FreshnessSeconds: 60 ** Content: "abcdefghij" * Send ContentObject by Face 1 * Socket OPEN for Face 1 * Protocol: 17 * Destination IP: 255.255.255.255 * Detination port: 1234 * Transmited ContentObject message * Opt 1 ended	LOG: Tue, 14/06/03, 00:00:07 * Received Interest (Opt 1) by Face 1 * Interest Message ** ContentName: "/ccnd/A/Fol2/D6C62F3FA327916424ADCA4A7A6C13B6" ** PPKD: D22B6FDCC90777469FF9815F7C0E0026 ** ChildSelector: Rightmost ** InterestLifeTime: 10 ** /ccnd/A/Fol2/D6C62F3FA327916424ADCA4A7A6C13B6 Correct format and SHA256 component. ** D22B6FDCC90777469FF9815F7C0E0026 Correct PPKD. * Build ContentObject from entry 7 * Entry 7 of CS: ** ContentName: /ccnd/A/Fol2/file4 ** Type: DATA ** StalenessBit: OFF ** CreationTimestamp: Tue, 14/06/03, 00:00:00 * FreshnessSeconds: 60 ** Last Time Used: Tue, 14/06/03, 00:00:08 * ContentObject Message ** Signarute: ** DigestAlgorithm: SHA256 ** Witness:IndSignature ** Signature Bits: 26359AA65D71E761B812839381848E39 ** ContentName: "/ccnd/A/Fol2/file4" ** SignedInfo: ** PPKD: D22B6FDCC90777469FF9815F7C0E0026 ** Timestamp: Tue, 14/06/03, 00:00:00 ** Type: DATA ** FreshnessSeconds: 60 ** Content: "abcdefghij#" * Send ContentObject by Face 1 * Socket OPEN for Face 1 * Protocol: 17 * Destination IP: 255.255.255.255 * Desination port: 1234 * Transmited ContentObject message * Opt 1 ended

B) Test 2 de operation1

- **Objetivo de la prueba**. Mostrar la ejecución del programa ante la llegada de un mensaje *Interest*.
- **Estado inicial:** Tabla PIT vacía y el ContentStore contiene las siguientes entradas.

```
* CONTENT STORE
* Entry 0: /ccnd/A/file1
* Entry 1: /ccnd/A/Fol1/Fol2/file2
* Entry 2: Empty
* Entry 3: /ccnd/B/fil4
* Entry 4: /ccnd/A/Fol2/file1
* Entry 5: /ccnd/A/Fol2/file3
* Entry 6: /ccnd/A/Fol2/file2
* Entry 7: /ccnd/A/Fol2/file4
* Entry 8: /ccnd/A/Fol2/file0
* Entry 9: /ccnd/A/file2
```

- **Ejecución:** Recepción de un mensaje *Interest* demandando el prefijo *"/ccnd/C"*. Dicho mensaje no puede ser satisfecho por el *ContentStore*. Se genera una nueva entrada en la tabla PIT y reenvía a través de la *Face* indicada por la tabla FIB. Posteriormente se vuelve a recibirse el mismo mensaje *Interest* y al estar registrado en la tabla PIT operation1 finaliza.

```
LOG: Tue, 14/06/03, 00:00:00
* Received Interest (Opt 1) by Face 1
* Interest Message
        ** ContentName:
"/ccnd/C/D28A6260569BB43F5BDA0CBC54CFFAF1"
        ** [...]
* ContentStore has not entries with the prefix "/ccnd/C"
* Opt 1 continues in PIT
* New entry in PIT
* Entry 0 of PIT:
        ** RequestedPrefix: /ccnd/C
        ** […]
        ** RequestedFace: 1
* Opt 1 continues in FIB
* Relay Interest by entry 3 of FIB.
* Entry 3 of FIB:
        ** Prefix: /ccnd/C
        ** StrategyID: 4
        ** Description: UNICAST_WASPMOTE_C
        ** OutputFaces: 4
* Send Interest by Faces 4
 * Socket OPEN for Face 4
         * Protocol: 17
         * Destination IP: 192.168.1.13
         * Detination port: 1234
 * Transmited Interest message
* Opt 1 ended
```

```
LOG: Tue, 14/06/03, 00:00:07
* Received Interest (Opt 1) by Face 1
* Interest Message
        ** ContentName:
"/ccnd/C/D28A6260569BB43F5BDA0CBC54CFFAF1"
        ** […]
* ContentStore has not entries with the prefix "/ccnd/C"
* Opt 1 continues in PIT
* "/ccnd/C" is registered in the PIT.
* Opt 1 ended
```

- **Resultado:** Una entrada en la tabla PIT asociada al prefijo "/ccnd/C".

6. 3. 2. Test de Operation2

A) Test 1 de operation2

- **Objetivo de la prueba.** Mostrar la ejecución del programa ante la llegada de un mensaje *ContentObject*.

- **Estado inicial:** Una entrada el PIT demandado el prefijo "/ccnd/C" y múltiples entradas en el *ContentStore*.

* PIT Table	* CONTENT STORE
* Entry 0 of PIT: /ccnd/C	* Entry 0: /ccnd/A/file1
* Entry 1 of PIT: Empty	* Entry 1: /ccnd/A/Fol1/Fol2/file2
* Entry 2 of PIT: Empty	* Entry 2: Empty
* Entry 3 of PIT: Empty	* Entry 3: /ccnd/B/fil4
* Entry 4 of PIT: Empty	* Entry 4: /ccnd/A/Fol2/file1
* Entry 5 of PIT: Empty	* Entry 5: /ccnd/A/Fol2/file3
* Entry 6 of PIT: Empty	* Entry 6: /ccnd/A/Fol2/file2
* Entry 7 of PIT: Empty	* Entry 7: /ccnd/A/Fol2/file4
* Entry 8 of PIT: Empty	* Entry 8: /ccnd/A/Fol2/file0
* Entry 9 of PIT: Empty	* Entry 9: /ccnd/A/file2

- **Ejecución:** Recibido un nuevo mensaje *ContentObject* con el *ContentName "/ccnd/C/fil1"*, se registra en el *ContentStore*. Se consulta la tabla PIT donde una entrada demandado el prefijo "/ccnd/C" puede ser satisfecha por el *ContentObject* recibido, se envía el *ContentObject* por la *Face* indicada en la tabla PIT y se elimina la entrada en la tabla PIT. Posteriormente, se vuelve a recibir el mismo *ContentObject* que renueva la entrada en el *ContentStore*.

LOG: Tue, 14/06/03, 00:00:07 * Received CO (Opt 2) by Face 1 * ContentObject Message ** Signarute: ** DigestAlgorithm: SHA256 ** Witness:IndSignature ** Signature Bits: E09322C1D6352B87626F467A289E00E9 ** ContentName: "/ccnd/C/file1" ** SignedInfo: ** PPKD: D22B6FDCC90777469FF9815F7C0E0026 ** Timestamp: Tue, 14/06/03, 00:00:07 ** Type: DATA ** FreshnessSeconds: 40 ** Content: "abcdefghij" ** E09322C1D6352B87626F467A289E00E9. Correct Signature. ** D22B6FDCC90777469FF9815F7C0E0026 Correct PPKD. * New entry in pos 2	LOG: Tue, 14/06/03, 00:00:13 * Received CO (Opt 2) by Face 1 * ContentObject Message ** Signarute: ** DigestAlgorithm: SHA256 ** Witness:IndSignature ** Signature Bits: E09322C1D6352B87626F467A289E00E9 ** ContentName: "/ccnd/C/file1" ** SignedInfo: ** PPKD: D22B6FDCC90777469FF9815F7C0E0026 ** Timestamp: Tue, 14/06/03, 00:00:12 ** Type: DATA ** FreshnessSeconds: 20 ** Content: "abcdefghij#" ** E09322C1D6352B87626F467A289E00E9. Correct Signature. ** D22B6FDCC90777469FF9815F7C0E0026 Correct PPKD. * Renewing entry 2

* Entry 2 of CS: ** ContentName: /ccnd/C/file1 ** Type: DATA ** StalenessBit: OFF ** CreationTimestamp: Tue, 14/06/03, 00:00:07 ** FreshnesSeconds: 40 ** Last Time Used: * Opt 2 continues in PIT * CO resolve entries in PIT: 0 * Entry 0 of PIT: ** RequestedPrefix: /ccnd/C ** CreationTimestamp: Tue, 14/06/03, 00:00:00 ** LastTimeTried: Tue, 14/06/03, 00:00:00 ** Tries: 1 ** InterestLifeTime: 120 ** RequestedFace 1 * Send CO by Face 1 * Socket OPEN for Face 1 * Protocol: 17 * Destination IP: 255.255.255.255 * Detination port: 1234 * Transmited ContentObject message * Deleting PIT's entry 0 * Updating CS's entry 2 * Entry 2 of CS: ** ContentName: /ccnd/C/file1 ** Type: DATA ** StalenessBit: OFF ** CreationTimestamp: Tue, 14/06/03, 00:00:07 ** FreshnessSeconds: 40 ** Last Time Used: Tue, 14/06/03, 00:00:12 * Opt 2 ended	* Entry 2 of CS: ** ContentName: /ccnd/C/file1 ** Type: DATA ** StalenessBit: OFF ** CreationTimestamp: Tue, 14/06/03, 00:00:12 ** FreshnessSeconds: 20 ** Last Time Used: * Opt 2 ended

B) Test 2 de operation2

- **Objetivo de la prueba.** Mostrar la resolución de múltiples peticiones pendientes ante la llegada de un mensaje *ContentObject*.

- **Estado inicial**: Múltiples entradas en la tabla PIT, dichas entradas fueron añadidas explícitamente para esta test de ahí el hecho de que estén repetidas. El contenido del ContentStore es indiferente para esta prueba.

```
* PIT Table
* Entry 0 of PIT: /
* Entry 1 of PIT: /ccnd
* Entry 2 of PIT: /ccnd/B
* Entry 3 of PIT: /
* Entry 4 of PIT: /ccnd
* Entry 5 of PIT: /ccnd/B
* Entry 6 of PIT: /
* Entry 7 of PIT: /ccnd
* Entry 8 of PIT: /ccnd/B
* Entry 9 of PIT: /
```

- **Ejecución:** Recibido un nuevo mensaje ContentObject asociado a *"/ccnd/C/file1"* que satisface múltiples entradas en la tabla PIT. Para facilitar la comprensión, se ha omitido parte del resultado original.

```
LOG: Tue, 14/06/03, 00:00:00
* Received CO (Opt 2) by Face 1
* ContentObject Message
        ** Signarute:  […]
        ** ContentName: "/ccnd/C/file1"
        ** SignedInfo: [...]
        ** [...]
* New entry in pos 2
* Entry 2 of CS:
        ** ContentName: /ccnd/C/file1
        ** [...]
* Opt 2 continues in PIT
* CO resolve entries in PIT: 0 1 3 4 6 7 9
* Send CO by Face 0[...]
* Deleting PIT's entry 0
* Send CO by Face 0[...]
* Deleting PÍT's entry 1
* Send CO by Face 0[...]
* Deleting PIT's entry 3
* Send CO by Face 0[...]
* Deleting PIT's entry 4
* Send CO by Face 0[...]
* Deleting PIT's entry 6
* Send CO by Face 0[...]
* Deleting PIT's entry 7
* Send CO by Face 0[...]
* Deleting PIT's entry 9
* Updating CS's entry 2
* Entry 2 of CS:
        ** ContentName: /ccnd/C/file1
        ** [...]
        ** Last Time Used: Tue, 14/06/03, 00:00:46
* Opt 2 ended
```

- **Resultado final:** Solo las entradas 2, 5 y 8 permanecen en la tabla PIT

```
* PIT Table
* Entry 0 of PIT: Empty
* Entry 1 of PIT: Empty
* Entry 2 of PIT: /ccnd/B
* Entry 3 of PIT: Empty
* Entry 4 of PIT: Empty
* Entry 5 of PIT: /ccnd/B
* Entry 6 of PIT: Empty
* Entry 7 of PIT: Empty
* Entry 8 of PIT: /ccnd/B
* Entry 9 of PIT: Empty
```

6. 3. 3. Test de Operation3

- **Objetivo de la prueba.** Mostrar la normal ejecución operation3, realizando reenvíos periódicos de lo mensajes pendiente y manteniendo la tabla PIT actualizada.

- **Estado inicial:** La tabla PIT contiene tres entradas pendientes con un tiempo de vida de 30 segundos cada una.

```
* PIT Table
* Entry 0 of PIT: /ccnd/C
* Entry 1 of PIT: /ccnd/C/file1
* Entry 2 of PIT: /ccnd/B
* Entry 3 of PIT: Empty
* Entry 4 of PIT: Empty
* Entry 5 of PIT: Empty
* Entry 6 of PIT: Empty
* Entry 7 of PIT: Empty
* Entry 8 of PIT: Empty
* Entry 9 of PIT: Empty
```

- **Ejecución:** Primer envío periódico a los 5 segundos (constante INTERVAL_TRIES) a través de la Face indicada por la tabla FIB. Pasados 30 segundos la entradas son borradas.

```
LOG: Tue, 14/06/03, 00:00:05
* Sending Pending Interest (Opt 3)
* Pending Interest in entry 0
* Opt 3 continues in FIB
* Relay Interest by entry 3 of FIB.
* Send Interest by Faces 4  [...]
* Pending Interest in entry 1
* Opt 3 continues in FIB
* Relay Interest by entry 3 of FIB.
* Send Interest by Faces 4  [...]
* Pending Interest in entry 2
* Opt 3 continues in FIB
* Relay Interest by entry 2 of FIB.
* Send Interest by Faces 3 [...]
```

```
LOG: Tue, 14/06/03, 00:00:33
* Sending Pending Interest (Opt 3)
* Deleting PIT's entry 0
* Deleting PIT's entry 1
* Deleting PIT's entry 2
* No pending Interest in PIT
* Opt 3 ended
```

- **Resultado final:** La tabla PIT queda completamente vacía.

7. Conclusiones y trabajos

Nuestra conclusión acerca del paradigma CCN es que es un enfoque de comunicaciones y distribución de contenidos muy interesante que resuelve ciertas deficiencias del modelo TCP/IP actual para ciertas aplicaciones.

- **Persistencia y carga de datos.** Los nodos TCP/IP emplean una gestión de memoria FIFO (*First Input First Output*) que elimina los paquetes de datos una vez transmitidos mientras que los contenidos son almacenados en máquinas concretas y dedicadas que soportan la carga de datos ante múltiples peticiones. Por otro lado, los nodos CCN emplean una gestión memoria LRU (*Least Recently Used*) o LFU (*Least Frecuently Used*) [21] que distribuye la carga de datos entre los todos nodos y la abstracción del concepto de localización permitiría convertir las redes en macro-sistemas de archivos.

- **Seguridad en las comunicaciones.** En una red TCP/IP, todos los nodos que intervienen en una comunicación son susceptibles de sufrir ataques, lo cual implica implementar mecanismos de seguridad en cada uno de ellos incrementando el tamaño y el tiempo de procesamiento de los mensajes. En el caso de CCN, cuando la seguridad es apoyada sobre los propios contenidos, no es útil dirigir estos ataques a una maquina concreta en la que hay una debilidad porque la seguridad está asociada a los contenidos, no a las máquinas.

- **Transmisión multi-trayecto libre de bucles.** A pesar de existencia de las transmisiones multicast, TCP/IP establece mayoritariamente comunicaciones punto a punto añadiendo mecanismos, cómo el campo TTL de IP, para evitar que un mensaje permanezca de forma indefinida en la red. CCN permite enviar un mismo bloque de datos por múltiples trayectorias mientras que con un control de flujo *uno -a-uno* se evitan los bucles.

Sin embargo, a pesar de sus ventajas, consideramos poco probable que el paradigma CCN sustituya completamente al modelo TCP/IP vigente, pero sí que puede ser aplicado en determinados escenarios, conviviendo o reemplazando el modelo TCP/IP en ciertas redes. En particular, las redes de sensores pueden ser un entorno de aplicación especialmente atractivo, porque el número de dispositivos es muy grande, y sin embargo las aplicaciones que usen la información generada están interesadas en contenidos, no en identificar el sensor o sensores que pueden proporcionarlos. El paradigma CCN permitiría, por lo tanto, crear un modelo de comunicación que simplificara crear aplicaciones sobre la red de sensores.

Por otro lado, en cuanto la plataforma Waspmote, nuestra conclusión es que la modularidad de dispositivo y un entorno de programación manejable hacen de esta plataforma una excelente elección a la hora de desplegar una red WSN. Sin embargo, Waspmote no está diseñado para distribuir contenidos en los términos que el proyecto CCNx establece ya que, tal y como, vimos en el apartado, algunas de las restricciones de los Waspmotes como la concurrencia, la gestión dinámica de memoria o la sincronización temporal de los dispositivos tienen difícil solución.

En cualquier caso, nuestro trabajo CCN abre una gama de nuevas líneas de desarrollo para futuros proyectos, tales como:

- Protocolos de enrutamiento basado en CCN.
- Implementación de los protocolos de gestión descritos por el proyecto CCNx [24] (ver apartado *2. 5. 1. 3. Reglas de estrategia y protocolos auxiliares*).
- Diferenciación de servicios en CCN, seleccionando distintas reglas de estrategia en base al tipo de flujo de información, audio-vídeo en tiempo real, transferencia de ficheros....
- Desarrollo de mecanismos de balanceo de carga para CCN.
- Hacer pruebas más amplias, involucrando una topología con un número grande de Waspmotes.
- Integrar la red de sensores basada en Waspmotes con nodos en la infraestructura que usarían comunicaciones basadas en CCN para interrogar a la red de sensores.

8. Anexos

Anexo I. Constantes, estructuras de datos y variables programables

A) Constantes principales

Constantes aplicadas a los ContentNames		
Nombre	**Valor**	**Descripción**
MAX_CN_COMP_SIZE	5	Número máximo de caracteres por componente.
MAX_CN_COMP_NUM	4	Número máximo de componentes por ContentName

Constantes aplicadas al cálculo SHA256		
Nombre	**Valor**	**Descripción**
SIZE_SHA256	16	Tamaño en bytes de los valores SHA256.
PUBLIC_KEY	"Libelium"	Clave publica para todos los contenidos y todos los publicadores de contenido.

Constantes de las funciones RTC		
Nombre	**Valor**	**Descripción**
SETUP_RTC	“14:06:03:03:00:00:00”	Valor al que se inicializa el reloj RTC a través de la función init_RTC() (Martes, 3 de junio de 2014 a las 00:00:00).
SIZE_TIMESTAMP	24	Número de caracteres, incluyendo la marca '\0', devuelto por la función getTime(), por ejemplo “Thu, 14/06/03, 00:00:00”.
SIZE_TYPED_TS	21	Número de caracteres, incluyendo la marca '\0', de una marca de tiempo en formato codificado, por ejemplo “14:06:03:03:00:00:00”.

Constantes aplicadas al mensaje Interest		
Nombre	**Valor**	**Descripción**
DF_CAN_ORDER	0	Orden canónico por defecto (*Leftmost Child*).
DF_INTEREST_LIFE	2	Tiempo de vida los mensajes *Interest* en la tabla PIT.

Constantes aplicadas al mensaje ContentObject		
Nombre	**Valor**	**Descripción**
NUM_TYPES	6	Cantidad de tipos de datos contenidos en un *ContentObject* (DATA, ENCR, GONE, KEY, LINK y NACK).
DF_FRESHNESS_SECONDS	60	Intervalo de tiempo en el que el bit de antigüedad permanecerá desactivado.
MAX_PAYLOAD	10	Tamaño máximo en bytes del contenido.

Constantes aplicadas a las Faces		
Nombre	**Valor**	**Descripción**
SIZE_FACE_LIST	4	Número de entradas de la lista de Faces.

Constantes aplicadas al ContentStore		
Nombre	**Valor**	**Descripción**
SIZE_FILESYSTEM	10	Número de entradas de un sistema de ficheros ficticio propio de cada nodo CCN.
SIZE_CONTENTSTORE	10	Número de entradas de la tabla ContentStore.
INTERVAL_NOT_USED	60	Intervalo de tiempo máximo que permanecerá una entrada del *ContentStore* con el bit de antigüedad activo antes de ser borrada.

Constantes aplicadas a la Tabla PIT		
Nombre	**Valor**	**Descripción**
SIZE_PIT	10	Número de entradas de la tabla PIT.
INTERVAL_TRIES	2	Intervalo de tiempo entre el reenvío de un mensaje Interest pendiente y el siguiente intento.

Constantes aplicadas a la Tabla FIB		
Nombre	**Valor**	**Descripción**
SIZE_FIB	5	Número de entradas de la tabla FIB.
NUM_OUTPUT_FACES	4	Número de Faces de salida por cada entrada de la FIB.

B) Estructuras de datos

Estructura	**Descripción**	
InterestMsg	Definición del mensaje *Interest*	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
Char *	contentname	40
uint8_t *	publisher_public_key_digest	16
uint8_t	child_selector	1
int	interest_life_time	2

Estructura	**Descripción**	
CO_Signature	Definición del campo Signature del ContentObject.	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
uint8_t	disgest_algorithm	1
uint8_t	witness	1
uint8_t *	signature_bits	16

Estructura	**Descripción**	
CO_SignedInfo	Definición del campo SignedInfo del ContentObject.	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
uint8_t *	publisher_public_key_digest	16
char *	timestamp	24
uint8_t *	type_co	3
int	freshness_seconds	2

Estructura	**Descripción**	
ContentObjectMsg	Definición del mensaje ContentObject	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
CO_Signature	signature	18
char *	contentname	24
CO_SignedInfo	signedinfo	45
char *	content	10

Estructura	**Descripción**	
Face	Definición de una Face	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
uint8_t	face_id	1
char *	host	16
uint8_t	proto	1
int	port	2

Estructura	**Descripción**	
ContentStoreEntry	Definición de una entrada en el ContentStore.	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
char *	contentname	24
uint8_t *	type_co	3
char	staleness_bit	1
char *	creation_timestamp	24
int	freshness_seconds	2
char *	last_time_used	24

Estructura	**Descripción**	
PitEntry	Definición de una entrada en la tabla PIT.	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
char *	requested_prefix	24
char *	creation_timestamp	24
char *	last_time_tried	24
uint8_t	tries	1
int	interest_life_time	2
uint8_t	requested_face	1

Estructura	**Descripción**	
FibEntry	Definición de una entrada en la tabla FIB.	
Tipo	**Nombre del Campo**	**Tamaño total (bytes)**
char *	prefix	24
uint8_t	strategy_rule_id	1
uint8_t *	output_faces	4
uint8_t	used_output_faces	1

C) Variables globales

Tipo	**Nombre**	**Tamaño (bytes)**	**Descripción**
InterestMsg	interest	59	Mensaje Interest.
ContentObjectMsg	content_object	97	Mensaje ContentObject.
ContentStoreEntry *	content_store	780	Tabla ContentStore.
PitEntry *	pit_table	760	Tabla PIT.
FibEntry *	fib_table	300	Tabla FIB.
Face *	list_faces	80	Lista de Faces.

Anexo II. Presupuesto

1.- Autor:	David Carrillo Sánchez
2.- Departamento:	Telemática

3.- Descripción del Proyecto:

- Titulo	Aplicación de Paradigma de Redes Centradas en Contenidos a Redes de Sensores
- Duración (meses)	**9**
Tasa de costes Indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):

Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Soto Campos, Ignacio		Ingeniero Senior	0,675	4.289,54	2.895,44	
Carrillo Sánchez, David		Ingeniero Junior	2,7	2.694,39	7.274,85	
					0,00	
					0,00	
		Hombres mes	**3,375**	**Total**	10.170,29	

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12

hombres mes (1575 horas)

Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS					
Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
HP EliteBook 8440p	325,49	100	9	60	48,82
PC Laboratorio	100,00	100	3	60	5,00
3xWaspmoteProV.1.2 + 3xModule Wifi Waspmote	474,00	75	3	60	17,78
Router inalámbrico ASUS RTN16	72,95	100	2	60	2,43
		100		60	0,00
					0,00
				Total	74,03

[d)] Fórmula de cálculo de la Amortización:

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS		
Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]		
Descripción	Empresa	Costes imputable
	Total	0,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	10.170
Amortización	74
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	2.049
Total	12.293

Anexo III. Reparto de tareas

Tarea	Horas de dedicadas
Documentación	72
Redacción del TFG	150
Implementación del prototipo	120
Pruebas finales sistema	10
Reuniones del equipo de trabajo	8
Total del TFG	360

REFERENCIAS

[1] Proyecto CCNx. Obtenido de http://www.ccnx.org/documentation/. Última revisión: Mar-2014.

[2] Palo Alto Research Center incorporated (PARC). Obtenido de http://www.parc.com/work/focus-area/content-centric-networking/. Última revisión: Ene-2014.

[3] Nikos Dimokas, Dimitrios Katsaros, Leandros Tassiulas,Yannis Manolopoulos (2011). "High performance, low complexity cooperative caching for wireless sensor networks".

[4] Pagina principal de Waspmote. Obtenido de http://www.libelium.com/development/waspmote/. Última revisión: Jun-2014.

[5] Named data networking. Obtenido de http://named-data.net/. Última revisión: Feb-2014.

[6] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. (2007) "A data-oriented (and beyond) network architecture".

[7] Zhong Ren, Mohamed A. Hail, Horst Hellbrück. (2013) "CCN-WSN – a lightweight, flexible Content-Centric Networking Protocol for Wireless Sensor Networks".

[8] D.R. Cheriton and M. Gritter. (2000) "Triad: A new next-generation internet architecture".

[9] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Laksh-inarayanan, and Ion Stoica. (2006) "Rofl: routing on flat labels".

[10] T. Berners-Lee (1998) RFC 2396. Internet Engineering Task Force.

[11] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard.(2012) "Networking named content". Communications of the ACM, Volume 55 Issue 1, January 2012.

[12] Proyecto CCNx, ContentName. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/NameConventions.html. Última revisión: Mar-2014.

[13] Proyecto CCNx, Orden Canónico. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/CanonicalOrder.html. Última revisión: Mar-2014.

[14] Manual de la función memcpy. Obtenido de http://www.cplusplus.com/reference/cstring/memcmp/ Última revisión: Jun-2014.

[15] Proyecto CCNx, Componente implícita SHA256. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/DigestComponent.html. Última revisión: Mar-2014.

[16] Proyecto CCNx, Mensaje Interest. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/InterestMessage.html. Última revisión: Mar-2014.

[17] Proyecto CCNx, Mensaje ContentObject. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/ContentObject.html. Última revisión: Mar-2014.

[18] Proyecto CCNx, Bit de antigüedad. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/Staleness.html. Última revisión: Mar-2014.

[19] Proyecto CCNx, Marcas de tiempo. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/Timestamp.html. Última revisión: Mar-2014.

[20] Proyecto CCNx, Firma digital. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/SignatureGeneration.html. Última revisión: Mar-2014.

[21] Definición de los algoritmos LFU y LRU. Obtenido de http://en.wikipedia.org/wiki/Cache_algorithms. . Última revisión: May-2014.

[22] Proyecto CCNx, Procesamiento de mensajes. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html. Última revisión: Mar-2014.

[23] Definición de protocolos de enrutamiento. Obtenido de http://docwiki.cisco.com/wiki/Internetworking_Technology_Handbook#_Routing. Última revisión: May-2014.

[24] Proyecto CCNx, Protocolos de gestión. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/Registration.html. Última revisión: Mar-2014.

[25] Proyecto CCNx, Algoritmos criptográficos. Obtenido de http://www.ccnx.org/releases/latest/doc/technical/CryptographicAlgorithms.html. Última revisión: Mar-2014

[26] Definición de redes WSN. http://es.wikipedia.org/wiki/Red_de_sensores. Última revisión: Feb-2014.

[27] Majid Bayani Abbasy, Gabriela Barrantes, Gabriela Marín (2011) “Time Delay Performance Analysis of Sensor Allocation Strategies on a WSN”

[28] Waspmote Technical Guide. Obtenido de http://www.libelium.com/es/development/waspmote/documentation/waspmote-technical-guide/ . Última revisión: Jun-2014

[29] Wifi Module Waspmote. Obtenido de http://www.libelium.com/development/waspmote/documentation/wifi-networking-guide/. Última revisión: Jun-2014

[30] Programing Guide Obtenido de http://www.libelium.com/development/waspmote/documentation/programming-guide/. Última revisión: Jun-2014

[31] Listado de plataformas WSN. Obtenido de http://wsn.oversigma.com/wiki/index.php?title=WSN_Platforms. Última revisión: Feb-2014

[32] Waspmote Datasheet. Obtenido de http://www.libelium.com/es/development/waspmote/documentation/waspmote-datasheet/. Última revisión: Jun-2014

[33] Manual de la función memcpy. Obtenido de http://www.cplusplus.com/reference/cstring/strcpy/. Última revisión: Jun-2014

[34] Código de ejemplo de recepción a través de un socket UDP. Obtenido de http://www.libelium.com/development/waspmote/examples/wifi-22-udp-client/. Última revisión: Jun-2014

[35] Códigos de ejemplo de Waspmote. Obtenido de http://www.libelium.com/development/waspmote/examples. Última revisión: Jun-2014

[36] Especificaciones Router Inalambrico ASUS RTN16. Obtenido de http://www.asus.com/Networking/RTN16/. Última revisión: Jun-2014

[37] Precio el Router Inalambrico ASUS RTN16. Obtenido de. http://www.amazon.es/ASUS-RT-N16-Router-gigabit-inal%C3%A1mbrico/dp/B0038734DM. Última revisión: Jun-2014

[38] Tutorial de Valgrind. Obtenido de http://cs.ecs.baylor.edu/~donahoo/tools/valgrind/ Última revisión: Jun-2014

BIBLIOGRAFÍA ADICIONAL

- Named Data Networking. http://en.wikipedia.org/wiki/Named_data_networking. Última revisión: Dic-2013
- Akyildiz, I., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). A survey on sensor networks. IEEE Communication magazine, 40(8)

- Karl, H., & Willig, A. (2005) Protocols and architectures for wireless sensor networks. New York: Wiley.
- 802.15.4 Networking Guide. Obtenido de http://www.libelium.com/development/waspmote/documentation/802-15-4-networking-guide/. Última revisión: Mar-2014
- ZigBee Networking Guide. Obtenido de http://www.libelium.com/development/waspmote/documentation/zigbee-networking-guide/. Última revisión: Mar-2014
- 868 Networking Guide. Obtenido de http://www.libelium.com/development/waspmote/documentation/868-networking-guide/. Última revisión: Mar-2014
- 900 Networking Guide. Obtenido de http://www.libelium.com/development/waspmote/documentation/900-networking-guide/. Última revisión: Mar-2014
- Bluetooth Networking Guide. Obtenido de http://www.libelium.com/development/waspmote/documentation/bluetooth-networking-guide/. Última revisión: Mar-2014
- GSM/GPRS Networking Guide. Obtenido de http://www.libelium.com/development/waspmote/documentation/gsmgprs-networking-guide/. Última revisión: Mar-2014
- Wireless Sensor Networks with Waspmote & Meshlium. Obtenido de http://downloads.libelium.com/documentation/wsn-waspmote_and_meshlium_eng.pdf. Última revisión: Mar-2014